UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Módulo biométrico de imágenes para reconocimiento facial de los usuarios

Autor/es

JAVIER SANTAMARÍA FORMOSO

Director/es

ANA ROMERO IBÁÑEZ

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Módulo Biométrico de imágenes para reconocimiento facial de los usuarios

Alumno:

Javier Santamaría Formoso

Tutores:

Ana Romero Ibáñez

Logroño, Junio,2017

Resumen

El presente Trabajo Fin de Grado trata sobre la creación de un servicio web de reconocimiento facial.

Los objetivos que se han cumplido son:

- Estudio de las técnicas de detección y reconocimiento facial.
- Creación de un entorno simulado de una plataforma web con el servicio de reconocimiento.
- Proceso de integración en una plataforma externa.

Para ello se han estudiado los algoritmos de reconocimiento facial que proporciona la librería OpenCV. Éstos son EigenFace, FisherFace y LBPHFace.

Estos tres algoritmos se han integrado en el servicio y se han hecho pruebas con el algoritmo LBPHFace.

De los resultados, hemos obtenido el número óptimo de imágenes de muestra por cada usuario, los rangos de acierto y fallo del algoritmo, y las condiciones de luz y posición para que el reconocimiento facial sea válido.

Abstract

This Final Grade Project is about creating a web service for facial recognition.

The objectives that have been fulfilled are:

- The study of techniques for detection and facial recognition.
- The creation of a simulated environment of a web platform with the integrated facial recognition service.
- The integration process in an external platform.

For this we have studied the facial recognition algorithms provided by the OpenCV library. These are EigenFace, FisherFace and LBPHFace.

From the results, we obtained the optimum number of sample images per user, the success and failure ranges of the algorithm and the light and position conditions for facial recognition to be valid.

Agradecimientos

Gracias a mi familia y a mis amigos por apoyarme en esta aventura que ha sido mi paso por la Universidad de La Rioja.

Gracias a ADR Formación y a todo su equipo por brindarme la oportunidad de trabajar con ellos y poder llevar a cabo este proyecto, y sobre todo por enseñarme lo que es trabajar en una gran familia.

Gracias a mi tutora Ana Romero por ayudarme en este trabajo con sus orientaciones y sus sugerencias las cuales han hecho posible el presente trabajo.

Índice

1.Introducción

1.1. Antecedentes

ADR Formación desarrolla proyectos de formación e-learning para sus clientes desde hace más de 15 años. Para ello, utilizan una plataforma de formación online de desarrollo y diseños propios en la que priman la sencillez y diferenciación.

Además, ofrecen un amplio catálogo de cursos creados a través de su Red de Talento conformada por expertos de diversas materias.

Alguna de las empresas que confían sus servicios de formación a ADR Formación son, entre otros, Cruz Roja, INECO, Diputación de Coruña y la universidad de Castilla-La Mancha.

ADR Formación quiere crear un servicio conectable a las plataformas que identifique a los usuarios a la hora de entrar en ellas mediante reconocimiento facial y comprobar así la autenticación de los usuarios.

1.2. Alcance

El objetivo del proyecto es la identificación de los usuarios en la plataforma a través de la captura de imágenes en tiempo real desde la webcam de su dispositivo.

Esto dará una medida más de seguridad en cuanto a la realización de los cursos y de los exámenes por parte de los estudiantes de la plataforma, asegurando que es el alumno quien está realmente detrás del dispositivo.

Para ello hará falta una 1º fase de recogida de imágenes en el registro del usuario, y una 2º fase de autenticación.

Nuestro proyecto estará dividido en 3 partes:

- La base de datos que contendrá los datos de los usuarios, los cursos a los que pertenece, las imágenes y los datos biométricos que serán puntos clave de las fotos tomadas.
- El servicio web que será el responsable de tratar las imágenes y analizar los datos biométricos de todas ellas utilizando machine learning.
- La interfaz web con la que interactuarán los usuarios.

1.3. Metodología

La metodología elegida para este proyecto, es un desarrollo en cascada, puesto que iremos desarrollando el proyecto en las distintas fases de análisis, diseño, implementación y pruebas.

1.4. Tecnologías

Para el desarrollo del proyecto hemos elegido crear la aplicación/servicio en Java. Esta decisión se basa principalmente en que ADR tiene su plataforma en PHP, pero la librería que necesitamos para poder crear el reconocimiento, OpenCV, de la que hablaremos más adelante, no se encuentra disponible para PHP. Por tanto crearemos un servicio al

que llamaremos a través de javascript con AJAX y JSON las cuales son tecnologías compatibles con PHP.

1.5. EDT

A continuación podemos ver el esquema de descomposición de tareas que vamos a tener en nuestro proyecto.

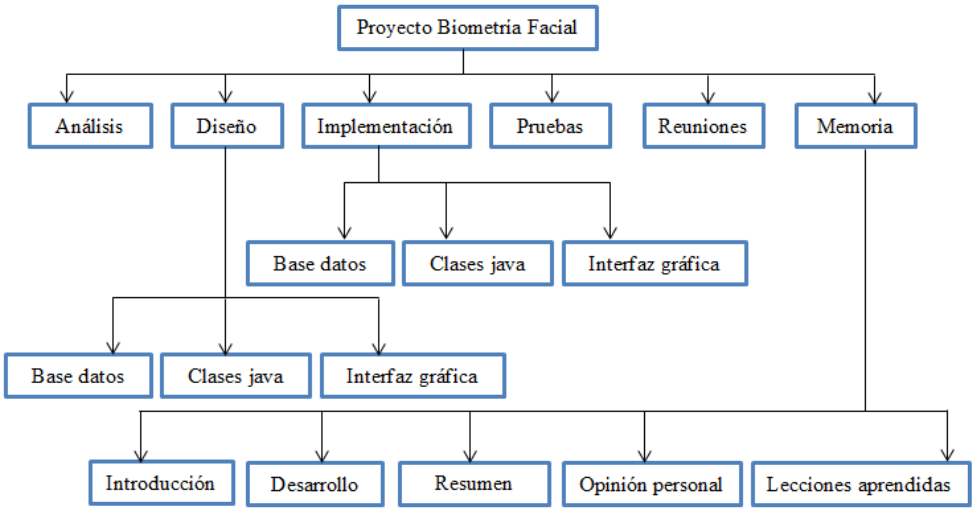

Ilustración 1: EDT

1.6. Tareas

a. Descripción

Análisis: analizaremos el problema que nos concierne a través de los requisitos funcionales de nuestra aplicación, con lo que después realizaremos el diseño oportuno.

Diseño: se desarrollará el diseño tanto de la base de datos, como de las clases java y de la interfaz gráfica que se utilizará.

Implementación: se implementará lo que se ha diseñado de forma que todo concuerde y se desarrollará el código que hará funcionar a la aplicación.

Pruebas: se desarrollarán las pruebas para saber que el software funciona óptimamente.

Reuniones: se tendrán reuniones tanto con el cliente como con el tutor del proyecto para dar los avances del mismo.

Memoria:

- Introducción: puesta en marcha del proyecto y planificar lo que vamos hacer en el mismo.
- Desarrollo: iremos desarrollando la memoria en la cual reflejaremos lo que se va realizando en el proyecto, explicando qué y cómo se hace.
- Resumen: se realizará una pequeña introducción que explique el motivo del proyecto y un breve resumen (en castellano y en inglés) que nos explique lo más destacado del proyecto.
- Opinión personal: en este apartado escribiremos nuestra opinión personal del

desarrollo del proyecto.

- Lecciones aprendidas y bibliografía: redactaremos un apartado donde se reflejen las cosas aprendidas durante la realización del proyecto. También redactaremos la Bibliografía con las referencias a los sitios en los que nos hemos apoyado para la realización del mismo.

b. Dependencias

Vamos a ver que tareas dependen de las otras:

Tarea		Dependencia
Desarrollo de la memoria	Planificación	----------
	Desarrollo	Depende de la realización de las tareas de análisis, diseño e implementación
	Introducción y resumen	----------
	Opinión personal	----------
	Lecciones aprendidas y bibliografía	Depende de la realización de las tareas de análisis, diseño e implementación
Reuniones	----------	----------
Pruebas	----------	Depende de que el servidor la BD y el cliente estén creados
Análisis	----------	----------
Diseño	Base de datos	Depende Análisis
	Clases Java	Depende del Diseño de la BD
	Interfaz gráfica	Depende Análisis
Implementación	Base de datos	Depende del Diseño de la BD
	Clases Java	Depende del Diseño de Clases Java
	Interfaz gráfica	Depende del Diseño de la Interfaz gráfica

Tabla 1: Dependencias de las tareas

c. Periodo de ejecución

Para nuestro Proyecto, nuestra planificación será de 16 semanas para 300 horas, las cuales vamos a dividir en semanas de 20 horas (5 días a 4 horas diarias).

En 14 de ellas se hará una jornada normal y en 2 de ellas se harán 8 y 12 horas respectivamente.

Las tareas que entran dentro de nuestra planificación son las del nivel inferior de cada apartado, las cuales se han visto en el diagrama EDT anterior.

Las fechas de planificación de las tareas por semanas son las siguientes:

- ***Análisis***: semana 2.
- ***Diseño:***
  - Base de datos: semana 2.
  - Clases Java: semanas 2 y 3.
  - Interfaz gráfica: semanas 3 y 4.
- ***Implementación***:

- Base de datos: semanas 4 y 5.
- Clases Java: semanas 5 a la 15.
- Interfaz gráfica: semanas 9 a la 15.

- ***Reuniones***: semanas pares.
- ***Pruebas***: Semanas 15 y 16.
- ***Desarrollo de la memoria***:
  - Planificación: semanas 1.
  - Desarrollo: semanas 2 a la 15.
  - Introducción y resumen: semanas 15 y 16.
  - Opinión personal: semana 14.
  - Lecciones aprendidas y Bibliografía: semana 15.

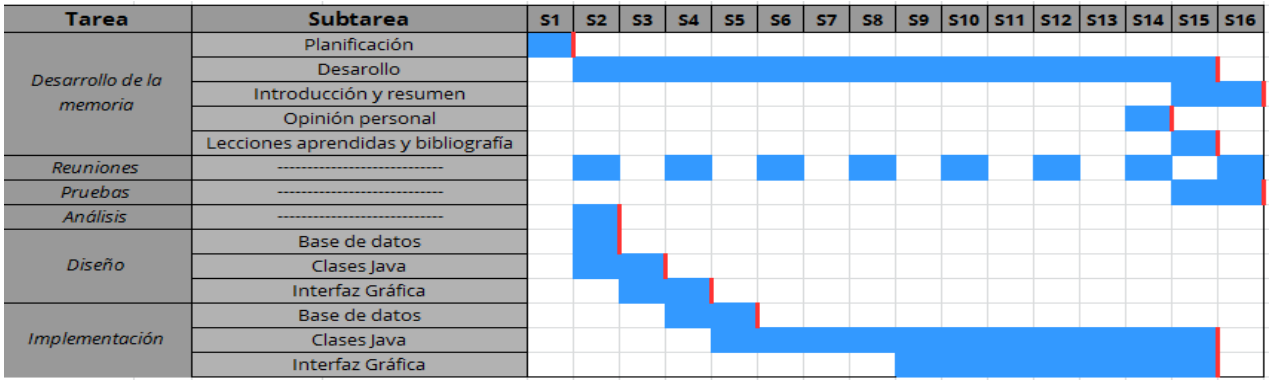

Ilustración 2: Diagrama de Gantt

d. Hitos

En el siguiente gráfico vemos los hitos más importantes de cada tarea. Básicamente reflejan las entregas/finalizaciones de cada tarea, y en cuanto a las reuniones nos indican cuando hay planificado realizar una.

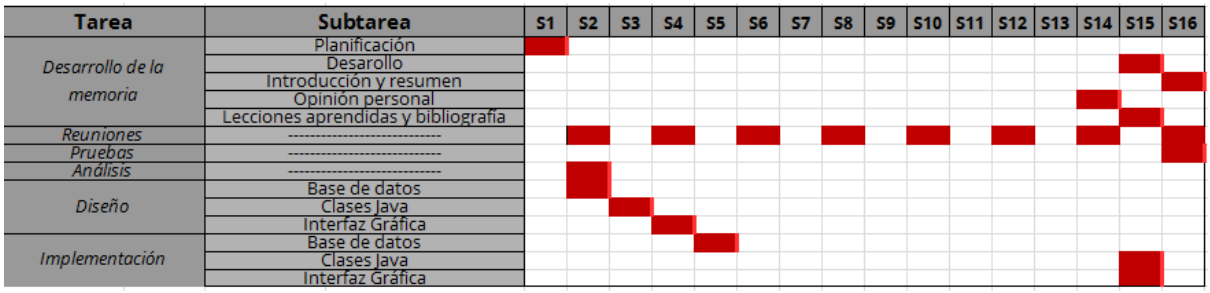

Ilustración 3: Hitos importantes de las tareas

1.7. Estimación dedicada a cada tarea

La planificación de horas de cada tarea que se refleja a continuación es la que podemos ver en el diagrama de la ilustración 2 pero diciendo las horas dedicadas durante ese periodo de tiempo.

Tarea	Horas
Total Memoria	**37**
Planificación	8
Desarrollo	22
Introducción y resumen	3
Opinión personal	2
Lecciones aprendidas y bibliografía	2
Reuniones	**8**
Pruebas	**10**
Análisis	**10**
Total Diseño	**33**
Diseño BD	5
Diseño Clases Java	20
Diseño Interfaz Gráfica	8
Total Implementación	**202**
Implementación BD	30
Implementación Clases Java	130
Implementación Interfaz Gráfica	42

Tabla 2: Estimación de horas por tarea

2. Análisis

2.1. Requisitos funcionales

Se quiere crear un servicio para el reconocimiento facial de los usuarios que pueden acceder a los cursos de formación de la plataforma.

Para ello el usuario tendrá que hacerse una serie de fotos para luego poder realizar la autenticación biométrica a través de ellas.

Por tanto, el usuario en su registro se tomará fotos en tiempo real con la webcam, y se enviarán estos datos al servicio para que trate las imágenes recogidas. Esto es necesario para poder realizar el registro completamente, puesto que si no se realizan las fotos, se cancelará el registro.

En el siguiente apartado vamos a analizar la funcionalidad que necesitamos para nuestra aplicación.

Para ello, primero vamos a analizar quiénes van a utilizar la aplicación y qué tareas o funcionalidades va a tener cada uno de estos usuarios.

En el siguiente esquema de clases de uso, podemos ver que tenemos 3 actores diferenciados que son, el Administrador, el Usuario Estándar y el Usuario Registrado.

Estos son necesarios puesto que antes de hacer una integración final vamos a recrear una plataforma completa como la de ADR. Es decir, de los casos de uso que se recogen a continuación, los que realmente entrarán en la integración con la plataforma son los relativos al reconocimiento facial y el Usuario Registrado.

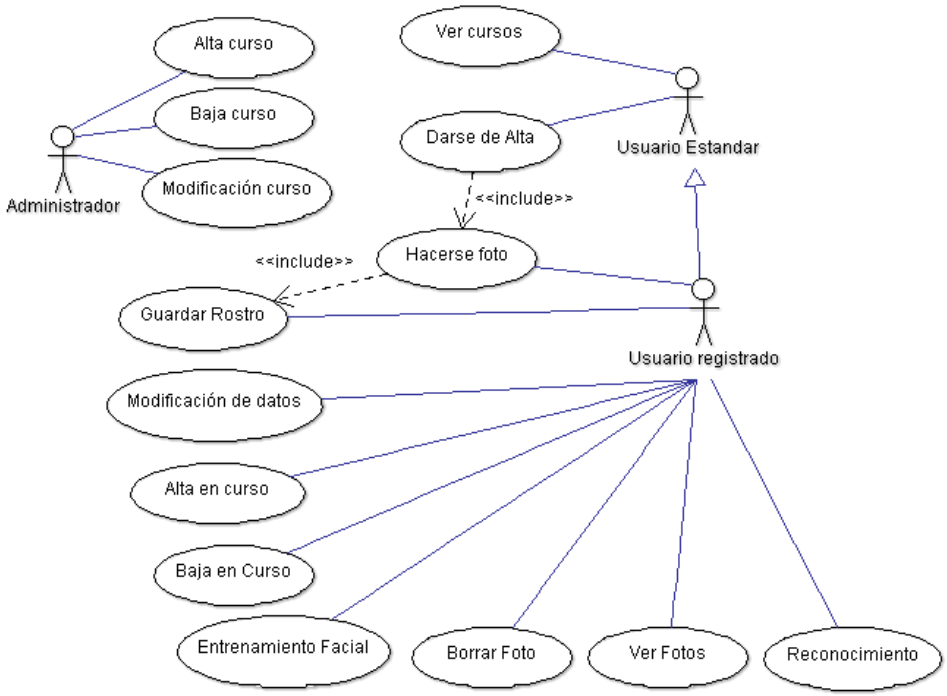

Ilustración 4: Casos de Uso

En el anterior esquema, todos los Casos de Uso del Usuario Registrado vienen precedidos de un login del que hemos prescindido en el esquema.

Para ver la funcionalidad que va a tener nuestra aplicación vamos a analizar los casos de uso que se ven reflejados en el esquema.

Empezaremos por los del actor Administrador, los cuales no entrarán en la integración final del servicio.

Nombre	Alta Curso
Descripción	Se dará de alta un curso en la base de datos
Actores	Administrador
Precondición	Ninguna
Flujo normal	
Acción de Actor	***Respuesta del Sistema***
1. Insertar nombre del curso 2. Insertar descripción del curso	3. Comprobar que no existe ese nombre de curso 4. Guardar curso en la base de datos
Flujo Alternativo	
	3.1. Si existe mandar un mensaje diciendo que ese curso está ya registrado
Postcondiciones	Ninguna

Tabla 3: Caso de uso Alta curso

Nombre	Modificación curso	
Descripción	Modificar los datos de un curso, es decir su nombre o su descripción	
Actores	Administrador	
Precondición	El curso debe de existir	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Seleccionar curso 2. Introducir nombre y descripción nueva		3. Modificación de los parámetros introducidos en la base de datos
Flujo Alternativo		
		3.1. Si hay algún parámetro en blanco no se modifica 3.2. Si el nombre nuevo introducido existe se envía un mensaje de error
Postcondiciones	Ninguna	

Tabla 4: Caso de Uso Modificación Curso

Nombre	Baja Curso	
Descripción	Eliminar un curso existente en la base de datos	
Actores	Administrador	
Precondición	El curso debe de existir	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Seleccionar curso		2. Eliminar de la base de datos
Flujo Alternativo		
		2.1. Si se produce algún error, no se elimina y se envía un mensaje de error
Postcondiciones	Ninguna	

Tabla 5: Caso de Uso Baja Curso

Los casos de uso que va a tener el Usuario Estándar van a ser dos, ver los cursos existentes y poder registrarse en la plataforma. Como en el caso del Administrador, este usuario y sus casos de uso son para simular la plataforma y no entrarán en la integración final del servicio.

Nombre	Ver cursos	
Descripción	Ver un listado con todos los cursos existentes y su descripción	
Actores	Usuario Estándar y Usuario Registrado	
Precondición	Debe existir algún curso	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Entrar en la sección de cursos 3. Entrar en la descripción de uno de los cursos		2. Devolver tabla con todos los cursos existentes en la plataforma. 4. Devolver la información sobre el curso seleccionado.
Flujo Alternativo		
Postcondiciones	Ninguna	

Tabla 6: Caso de Uso Ver cursos

Nombre	Darse de Alta	
Descripción	Registrarse en la plataforma para acceder a más contenidos	
Actores	Usuario Estándar	
Precondición		
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Insertar su información		2. Si es correcta dar de alta en la base de datos 3. Abrir sesión del usuario.
Flujo Alternativo		
		2.1. Si existe en la base de datos o los campos rellenados son incorrectos, devuelve un mensaje de error
Postcondiciones	El usuario está en la sesión, se ha convertido en usuario Registrado	

Tabla 7: Caso de Uso Darse de Alta

Por último vamos a ver los casos de uso que va a tener nuestro Usuario Registrado:

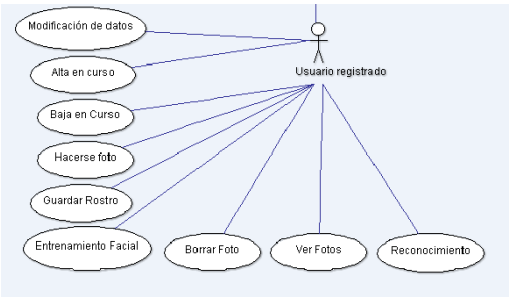

Ilustración 5: Casos de Uso del Usuario Registrado

Éste es el usuario que más casos de uso y funcionalidad tiene puesto que es él quien queremos que realice los procesos de reconocimiento facial. Éste, será quien va a tener que identificarse en la plataforma y pasar el nivel de seguridad que proporciona el reconocimiento facial. Para ello tendrá que pasar por 2 fases, una de recogida de imágenes con extracción de parámetros biométricos y la propia fase de reconocimiento.

Todos los casos de uso vistos en la *Ilustración 5* se detallan a continuación, pero los casos de uso Alta en curso, Baja en Curso y Modificación de datos son, como en los anteriores usuarios, métodos para la simulación del funcionamiento de la plataforma.

Nombre	Modificación de datos	
Descripción	El usuario puede modificar sus datos personales	
Actores	Usuario Registrado	
Precondición	El usuario debe de estar en la sesión	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Introducir datos nuevos		2. Si son correctos, modificarlos en la base de datos
Flujo Alternativo		
		2.1. Si hay un parámetro incorrecto o existe el nombre que se quiere modificar, enviar mensaje de error
Postcondiciones	Datos de la sesión modificados con los nuevos datos	

Tabla 8 : Caso de uso Modificación de Datos

Nombre	Alta en curso

Descripción	Matriculación de un usuario en uno de los cursos que existen	
Actores	Usuario Registrado	
Precondición	El usuario y el curso deben existir	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Accede a la lista de cursos 3. Selecciona el curso deseado		2. Devuelve todos los cursos que existen en la plataforma 4. Se da de alta al usuario en el curso
Flujo Alternativo		
		4.1. Si el usuario ya está matriculado no se realiza ninguna operación solo se devuelve a su página de inicio
Postcondiciones	Ninguna	

Tabla 9: Caso de Uso Alta en Curso

Nombre	Baja en curso	
Descripción	El usuario se sale de un curso en el que está matriculado	
Actores	Usuario Registrado	
Precondición	El usuario debe estar matriculado en algún curso	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
2. Se selecciona el curso en el cual se quiere dar la baja		1. Se devuelven todos los cursos en los que el usuario está matriculado 3. Se da de baja el usuario en ese curso
Flujo Alternativo		
		3.1. Si surge algún error se cancela la baja y se envía un mensaje de error.

Tabla 10: Caso de Uso Baja en Curso

Para el siguiente Caso de Uso, hay que decir que solamente se realiza cuando se hace el registro del usuario, y si no se pueden tomar las fotos se cancela el registro.

Nombre	Hacerse foto	
Descripción	Un usuario deberá hacerse una foto para posteriormente guardar el rostro que se detecte en ella	
Actores	Usuario Registrado	
Precondición	Usuario en sesión y acceso a webcam	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Acceder a la zona de captura de fotos 2. Pulsar botón para realizar foto		3. Se crea una instantánea captada por la cámara web y se pone en el parámetro que enviaremos al servidor para tratarla.
Flujo Alternativo		
		1.1. Si no se puede acceder a la webcam se notifica con un mensaje de error
Postcondiciones	Ninguna	

Tabla 11: Caso de Uso Hacerse Foto

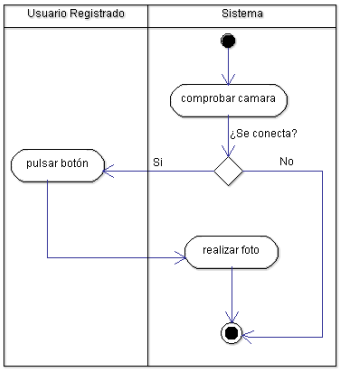

Ilustración 6: Diagrama de flujo Hacerse Foto

Nombre	Guardar Rostro
Descripción	En la foto realizada anteriormente se detecta un rostro y se guarda
Actores	Usuario registrado
Precondición	Imagen realizada por la webcam
Flujo normal	
Acción de Actor	**Respuesta del Sistema**
1. Envío de imagen realizada	2. Se detecta el rostro que hay en la imagen 3. Se recorta el rostro detectado 4. Se pasa a escala de grises 5. Se redimensiona a un tamaño específico 6. Se guarda la foto en el servidor 7. Se registra una entrada en la base de datos
Flujo Alternativo	
	2.1. Si no se detecta un rostro se envía un mensaje solicitando el envío de una nueva foto.
Postcondiciones	Ninguna

Tabla 12: Caso de Uso Guardar Rostro

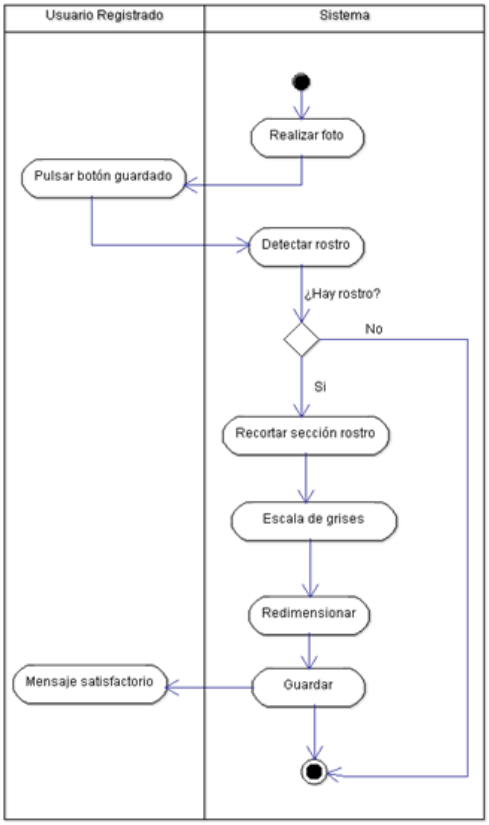

Ilustración 7: Diagrama de Flujo Guardar Rostro

El siguiente caso de uso, es dirigido por el usuario, aunque también funcionará de manera automática en algunos casos.

Nombre	Entrenamiento Facial
Descripción	Se recogen los datos biométricos de las imágenes que se han tomado del usuario en un archivo
Actores	Usuario Registrado
Precondición	El usuario debe tener fotos para sacar los datos biométricos de éstas
Flujo normal	
Acción de Actor	**Respuesta del Sistema**
1. Se accede a la zona de toma de imágenes 2. Se pulsa sobre el botón de entrenamiento facial	3. Se extraen los datos biométricos de las fotos que tiene el usuario 4. Se guardan todos esos datos en un fichero xml que será el que nos de la información para realizar el reconocimiento 5. Se registra una entrada en la base de datos
Flujo Alternativo	
	4.1. Si existe el fichero, se crea uno temporal para comprobar que tiene toda la información. 4.2. Si no la tiene se elimina el fichero antiguo y se guarda el temporal como persistente y se modifica el registro en la Base de datos.
Postcondiciones	Ninguna

Tabla 13: Caso de Uso Entrenamiento Facial

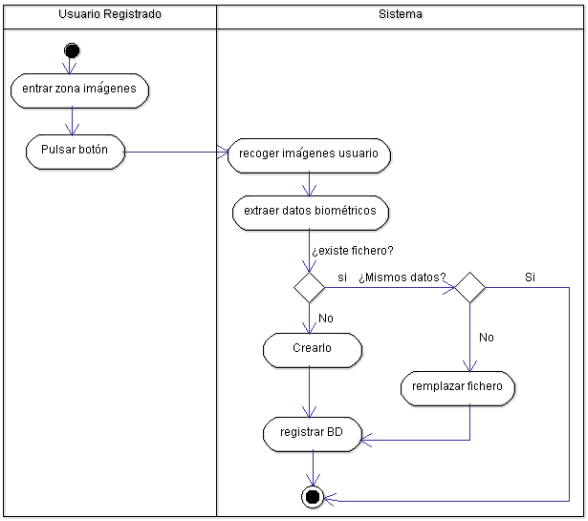

Ilustración 8: Diagrama de Flujo Entrenamiento Facial

Nombre	Ver fotos	
Descripción	Se mostrarán todas las fotos guardadas del usuario	
Actores	Usuario Registrado	
Precondición	El usuario debe tener alguna foto	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Entrar en la zona de usuario 2. Pinchar sobre un botón para ver las fotos 4. Se muestran las imágenes		3. Devolver la lista de imágenes que tiene el usuario
Flujo Alternativo		
4.1. Si no hay imágenes se muestra un mensaje.		
Postcondiciones	Ninguna	

Tabla 14: Caso de Uso Ver Fotos

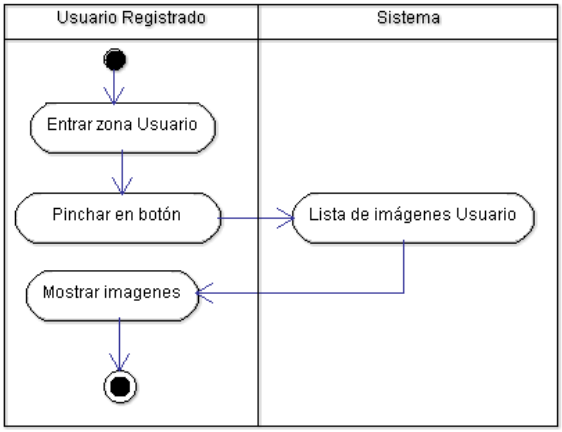

Ilustración 9: Diagrama de flujo Ver Fotos

Nombre	Borrar foto
Descripción	Borrado de una de las fotos del usuario
Actores	Usuario Registrado
Precondición	Debe existir al menos una foto del usuario
Flujo normal	
Acción de Actor	**Respuesta del Sistema**
1. Pulsar el botón de ver fotos 3. Seleccionar la imagen 4. Pulsar botón de borrado	2. Devolver las imágenes del usuario 5. Se coge el nombre de la imagen que se ha seleccionado y se borra de la base de datos. 6. Se elimina la imagen física del servidor
Flujo Alternativo	
	5.1. Si hay algún problema se envía un mensaje de error.
Postcondiciones	Ninguna

Tabla 15: Caso de Uso Borrar imagen

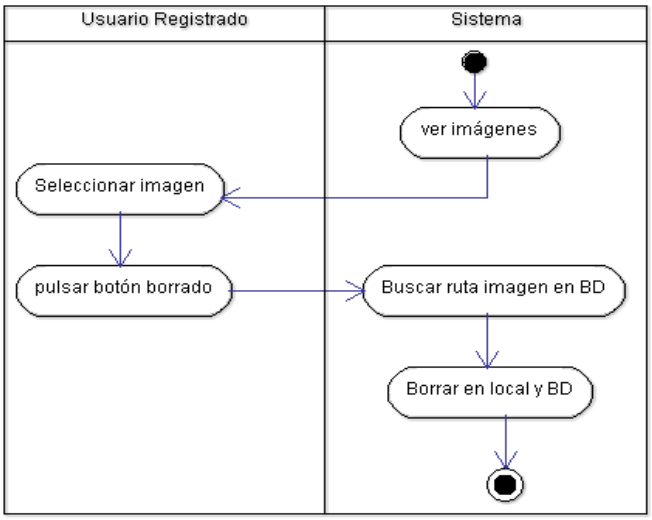

Ilustración 10: Diagrama de Flujo Borrar Imagen

Cuando queremos utilizar el reconocimiento, es cuando queremos entrar al perfil del usuario y a sus datos, es decir después de loguearnos haremos el reconocimiento facial para saber que es la persona adecuada y darle acceso. Si el resultado que vemos es el adecuado, nos dejará pasar.

Nombre	Reconocimiento	
Descripción	A partir de una imagen tomada se reconoce el rostro de la persona	
Actores	Usuario Registrado	
Precondición	Debe existir un fichero de entrenamiento con datos biométricos del usuario	
Flujo normal		
Acción de Actor		**Respuesta del Sistema**
1. Realizar foto 7. Ver el resultado del algoritmo utilizado		2. Detectar, recortar y almacenar en temporal el rostro de la imagen. 3. Extraer los datos biométricos 4. Compararlos con los del fichero 5. Devolver la máxima coincidencia 6. Eliminar imagen temporal
Flujo Alternativo		
7.1. Si su coincidencia es muy baja se devuelve un mensaje diciendo que el usuario no es el adecuado.		
Postcondiciones	Ninguna	

Tabla 16: Caso de Uso Reconocimiento

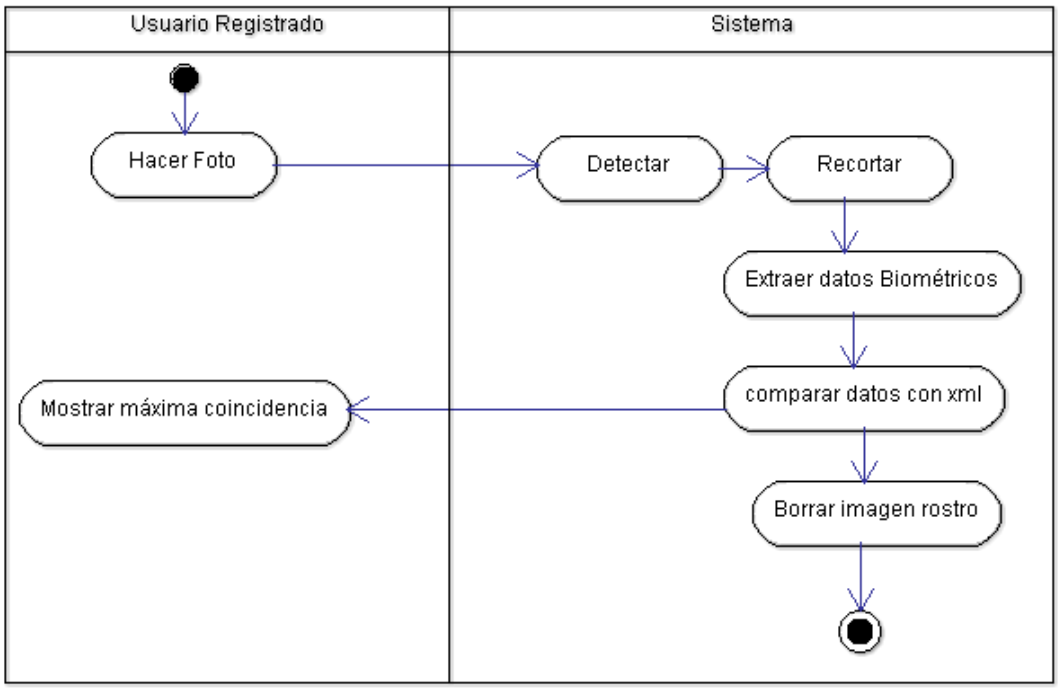

Ilustración 11: Diagrama de Flujo Reconocimiento

2.2. Requisitos no funcionales

Vamos a agrupar los requisitos en 4 modalidades: usabilidad, seguridad, multiplataforma y rendimiento.

- **Usabilidad**: debe de ser una aplicación intuitiva para el usuario, que sea fácil de manejar sin tener grandes conocimientos.
- **Seguridad**: la entrada al sistema estará bajo contraseña cifrada. Además el reconocimiento facial es una segunda medida de seguridad.
- **Multiplataforma**: se debe poder acceder desde cualquier dispositivo a través del navegador. Es un servicio que se llama a través de AJAX y por tanto todos los navegadores deberían soportarlo.
- **Rendimiento**: el sistema debe soportar el manejo de grandes cantidades de información, puesto que tratará los datos biométricos de los usuarios registrados en las plataformas de ADR.

3. Diseño

3.1. Base de datos

a. Requisitos de datos

Se quiere crear un servicio para el reconocimiento facial de los usuarios que pueden acceder a los cursos de formación de la plataforma.

Para ello el usuario tendrá que hacerse una serie de fotos para luego poder realizar la autenticación biométrica a través de ellas.

Por tanto, el usuario en su registro, y posteriormente, se tomará fotos en tiempo real con la webcam, y se enviarán estos datos al servicio para que trate las imágenes recogidas.

Cuando el usuario se registre, se guardará su identificador, su nombre, su constraseña y, si se ha hecho fotos, las imágenes que le identifican. De las imágenes se guardará su identificador, su nombre y la imagen codificada. Cabe decir que las imágenes solamente pertenecerán a un usuario en concreto, no se puede compartir una para varios usuarios.

Nuestros cursos están registrados con un identificador, un nombre y una pequeña descripción del mismo.

Para que el servicio de identificación biométrica funcione, hay que recoger de las fotos sus parámetros biométricos, es decir, esta será la parte de entrenamiento. El fichero donde se recogen todos estos parámetros funciona por usuario, es decir, cada usuario tendrá un fichero XML con los datos biométricos de las imágenes que le pertenecen.

Cuando tenga ese fichero, lo que haremos será guardarlo con un identificador, la ruta del fichero y un resumen del mismo.

b. Modelo Entidad/Relación

En la siguiente figura podemos ver el modelo Entidad/Relación que se ha obtenido a partir de los requisitos de datos anteriormente mencionados.

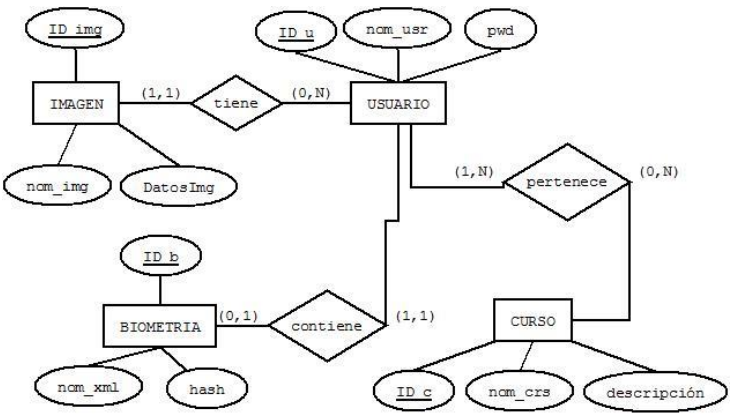

Ilustración 12: Modelo E/R

c. Diseño lógico

Vamos a crear el diseño lógico de las tablas que entran dentro de nuestro esquema E/R y comprobar así si resulta alguna tabla nueva a partir de las relaciones que existen entre las entidades.

Usuario

ID_u	nom_usr	pwd

Curso

ID_c	nom_crs	descripción

usr_crs

ID_u	ID_c
CE: Usuario	CE: Curso

Imagen

ID_img	ID_u	nom_img	DatosImg
	CE: Usuario		

Biometría

ID_b	ID_u	nom_xml	contenido
	CE: Usuario		

d. Descripción de las tablas

- **USUARIO**: esta tabla recogerá el nombre de usuario y su contraseña. Además va a contar con un identificador numérico que es único, no se puede repetir. Además también haremos que el nombre de usuario sea único, con lo que nos aseguraremos de que no hay 2 usuarios con el mismo nombre.
- **CURSO**: ésta recogerá los datos de los cursos en los que el alumno se puede matricular y en los cuales estará disponible la biometría. Los datos recogidos son, el nombre y la descripción del curso, además de un identificador numérico que lo identifique.
- **USR_CRS**: esta tabla será la que recoja la relación que hay entre los usuarios y los cursos. En ella estarán relacionados como clave primaria, los 2 atributos claves de las anteriores tablas. Con ella, podremos saber qué usuarios pertenecen a un curso en específico o en qué cursos está matriculado un usuario en concreto.
- **IMAGEN**: esta tabla contendrá las imágenes que formarán parte del proceso de biometría. Guardaremos su ruta (nom_img), un identificador, el usuario al que pertenece la imagen, puesto que cada una solo puede pertenecer a uno, y la propia imagen como atributo (formato Blob).
- **BIOMETRIA**: en esta tabla tendremos los datos de los ficheros xml de biometría. En ella se guardará un identificador numérico para el fichero, el identificador del Usuario al que pertenece esa biometría, la ruta donde se encuentra el fichero (nom_xml) y el hash en md5 del fichero. La razón del guardar el hash es que los

ficheros, cada vez que recogemos una imagen nueva, cambian los datos. En estos ficheros XML se recogerán todos los datos de las imágenes que se recogen en el entrenamiento y que sirven de referencia para el posterior reconocimiento facial.

e. Normalización:

1) Todas las tablas cumplen con la ***1ª Forma Normal***:

Solo hay atributos monovaluados.

2) Todas las tablas cumplen con la ***2ª Forma Normal***:

Todo atributo no primo depende funcionalmente de manera total de toda clave de la tabla.

3) Todas las tablas cumplen con la ***3ª Forma Normal***:

Para toda DF X → Y de la tabla: X es superclave o bien Y es atributo primo.

4) Todas las tablas cumplen con la ***Forma Normal de Boyce-Codd***:

Para toda DF X → Y de R X es superclave.

3.2. Clases Java

El diseño de las clases Java, lo vamos a tener dividido en 3 capas: persistencia, lógica de negocio y aplicación, aunque esta estará más relacionada con el diseño de la interfaz gráfica.

Además vamos a tener también un modelo de datos que se va a adecuar al diseño de la base de datos, con las clases Usuario, Curso, Biometría, Imagen y Usr_Crs.

a. Diseño capa de persistencia

Esta capa, va a contener las clases que conectarán con nuestra base de datos. Para ello crearemos una clase por cada tabla, para tener los métodos que acceden, insertan, modifican y borran valores en éstas.

También tendremos una clase que creará la conexión a la base de datos.

Es decir, el esquema será el siguiente:

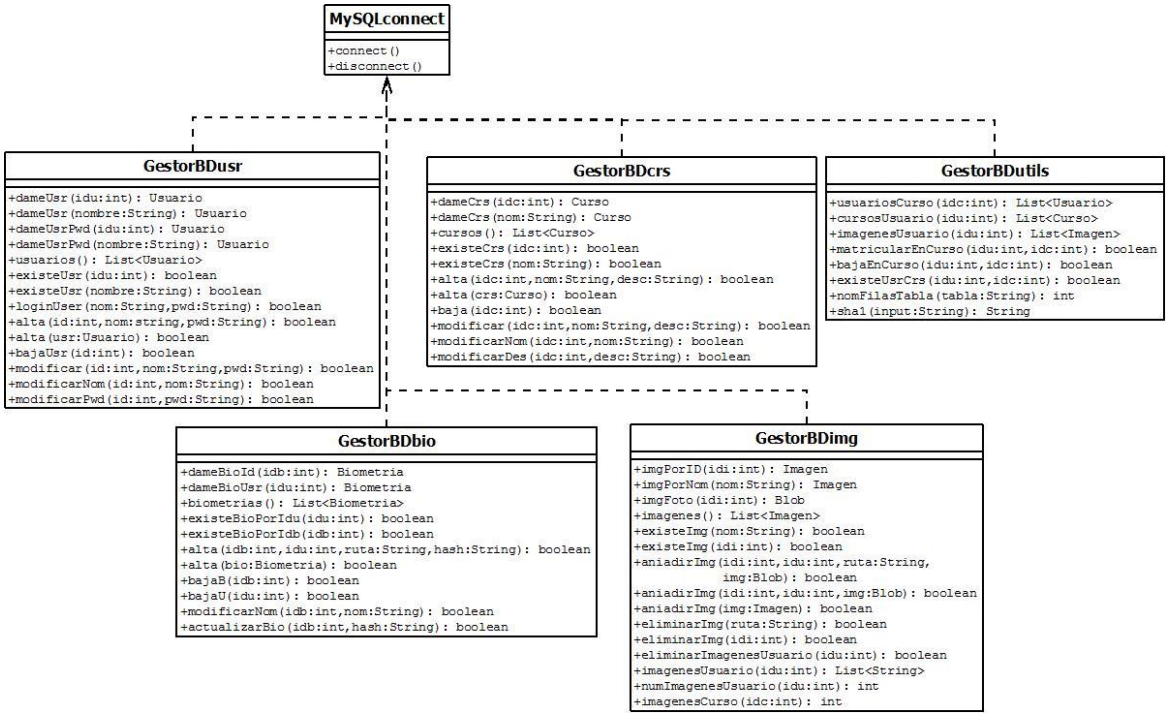

Ilustración 13: Clases java Persistencia

b. Diseño modelo de datos

Las clases del modelo de datos seguirán el siguiente esquema:

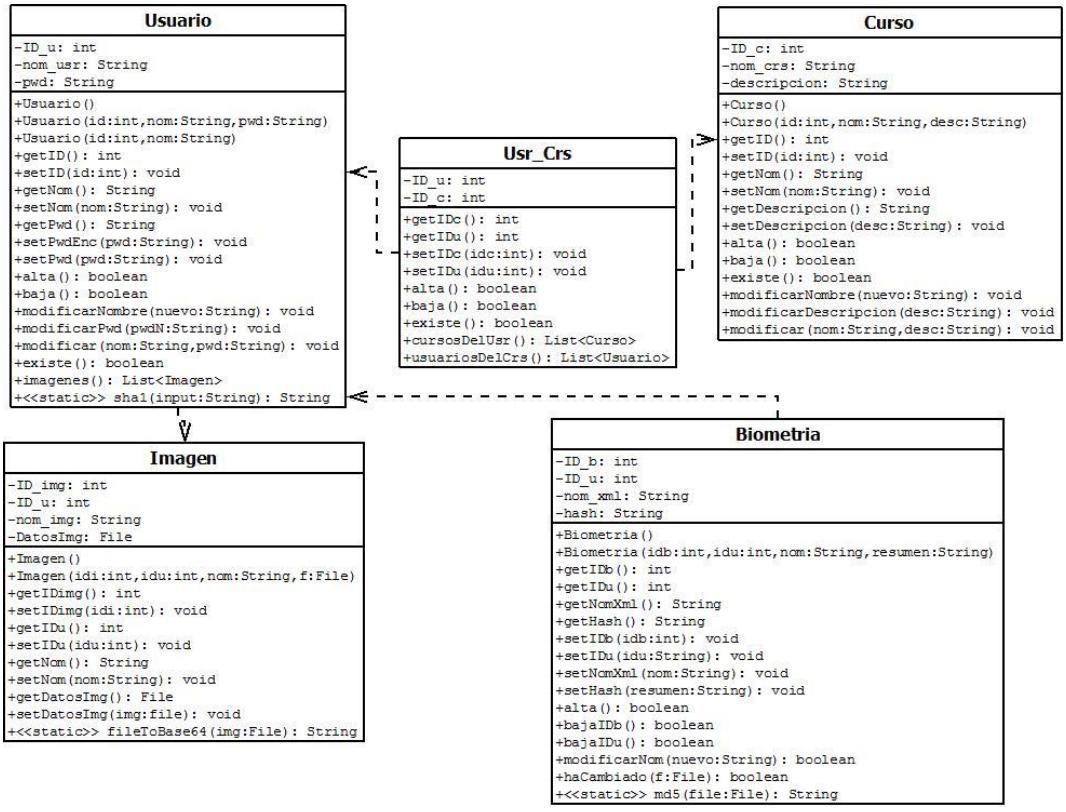

Ilustración 14: Clases del Modelo de Datos

c. Diseño capas lógica de negocio y aplicación

En la capa de Lógica de Negocio vamos a tener 3 paquetes diferenciados que son:

-Servlets: contendrá las clases que nos darán funcionalidad a la web haciéndola dinámica.

-logicaNegocio: en este paquete tendremos las clases que realizarán los procesos de biometría, tanto el crear las imágenes adecuadas para ella, como realizar el entrenamiento y el reconocimiento.

-api.ajax: este paquete contendrá las clases que nos darán una respuesta a la página web, a través de JQuery, Ajax y Json.

Para la capa de aplicación, hay que comentar que se hará a través de jsp, html y javascript. Por tanto a la hora de implementar la interfaz gráfica de la web, se utilizarán estas tecnologías para poder dar funcionalidad a la misma.

3.3. Interfaz gráfica

La interfaz con la que interactuarán nuestros usuarios se va a dividir en 3 zonas:

- La primera será la que puedan ver todos los usuarios, tanto registrados como los que no, que tendrá la página inicial, el catálogo de todos los cursos, la página de descripción de cada curso y la página de registro de usuarios.

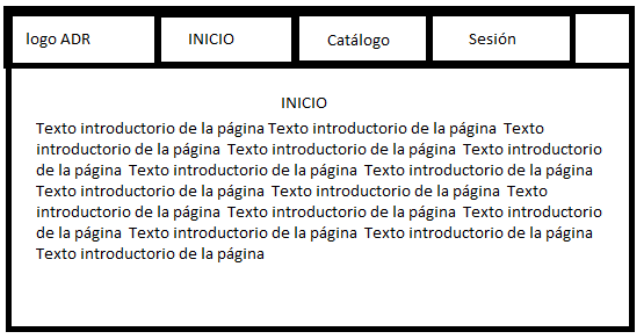

Ilustración 15: Diseño índex

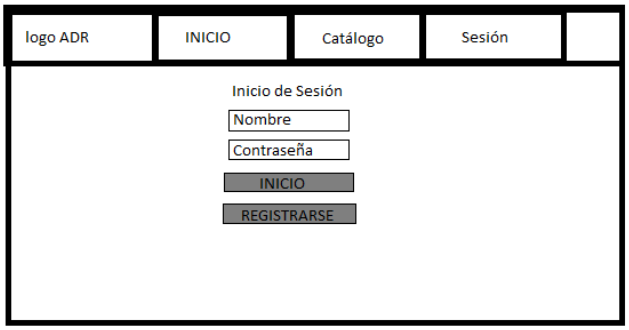

Ilustración 16: Diseño inicio sesión

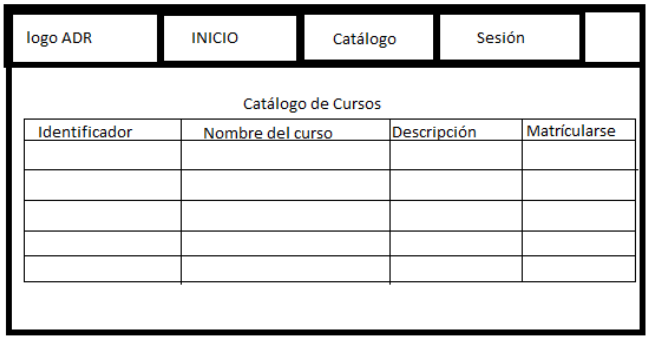

Ilustración 17: Diseño catálogo

- La segunda, para los registrados, además de lo anterior, mostrará su página personal donde podrá ver sus datos, los cursos en los que está matriculado y la sección de entrenamiento/reconocimiento facial.

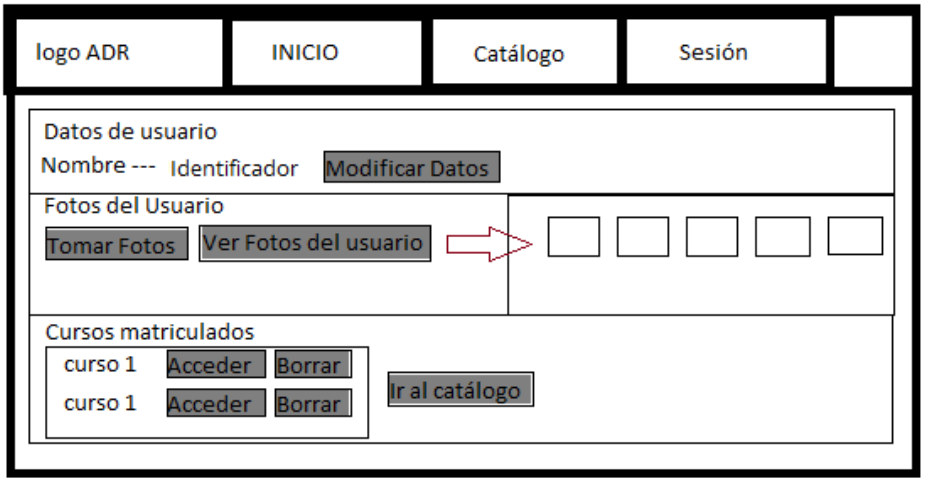

Ilustración 18: Diseño página usuario

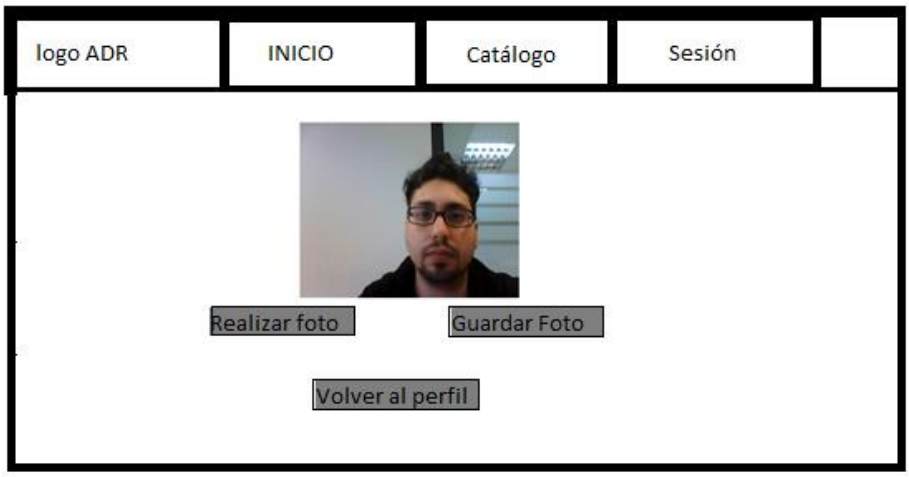

Ilustración 19: Diseño recogida de imágenes

Ilustración 20: Diseño página de reconocimiento facial

Esta última imagen puede variar en lo que se refiere al texto del resultado cuando hayamos hecho la implementación.

- Y la tercera para el usuario Administrador, que cuando entre en su perfil, en vez de tener una página personal como el resto de usuarios, tendrá una página donde podrá administrar los cursos, tanto dar de alta uno nuevo, dar de baja alguno existente y poder modificar los datos de estos.

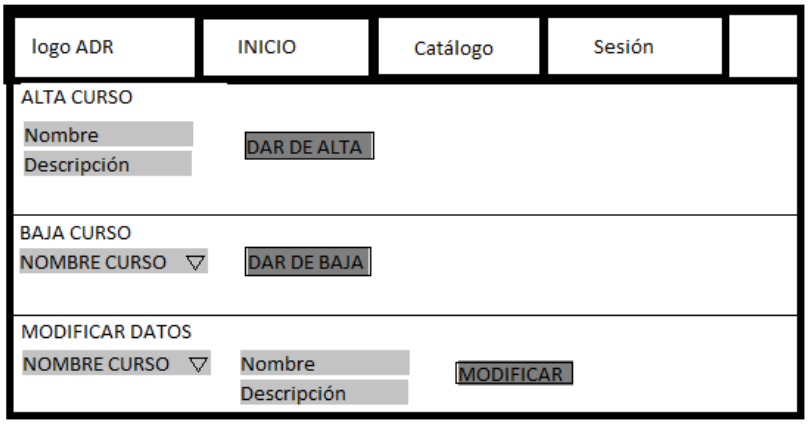

Ilustración 21: Diseño administración de cursos

4. Implementación

4.1. Especificaciones técnicas

Para desarrollar nuestro proyecto vamos a necesitar herramientas software distintas. Estas serán las siguientes:

- Hardware:
  - Para el desarrollo de la aplicación vamos a utilizar una máquina Ubuntu 16.04 para poder utilizar las herramientas gráficas.
  - El servidor donde alojaremos finalmente el servicio, es un CentOS 7 de Amazon sin capa gráfica.
- Software:
  - Base de datos: contaremos con una base de datos MySQL a la cual se accederá por consola puesto que es factible la implementación a partir de la terminal porque no tendremos muchas tablas.
  - Servicio web: para desarrollar la aplicación web lo haremos a través de NetBeans y un servidor Apache Tomcat.
  - El lenguaje de programación elegido será Java. Además utilizaremos otros lenguajes o tecnologías como son: HTML, JDBC, JavaScript, JSON, AJAX y JQuery.
  - Para poder crear el servicio de biometría en la parte del reconocimiento elegiremos la biblioteca OpenCV de la cual se hablará más adelante.
  - En cuanto a la capa gráfica haremos uso de Bootstrap, una herramienta que nos permite dar un diseño más profesional a nuestra web a través de css ya creados y de una rápida implementación.
  - Otras herramientas utilizadas: ArgoUML, DIA, Microsoft Word y Microsoft Excel.

4.2. Base de datos

Para implementar la base de datos, primero hay que elegir en que plataforma la vamos a montar. En nuestro caso nos hemos decantado por MySQL puesto que el servidor en el que montaremos la infraestructura completa será un CentOS 7 sin capa gráfica, y para la capa de persistencia utilizaremos JDBC para crear la conexión a la Base de datos.

4.3. Lógica de negocio

Lo primero que tenemos que analizar es cómo vamos a realizar el reconocimiento facial. Para ello existen algoritmos matemáticos que nos ayudarán con el proceso, pero antes de ello necesitamos obtener rostros de las imágenes, y para ello vamos a basarnos en el algoritmo Viola-Jones.

4.3.1. Algoritmo Viola-Jones

Este algoritmo se basa en tres conceptos.

- El primero de ellos es conocido como "Imagen Integral", que permite que las características tipo “haar” utilizadas por este detector se puedan calcular muy rápido.
- El segundo es un algoritmo de aprendizaje automático, "Adaboost", que selecciona sólo las características importantes de todo el conjunto.
- El tercer concepto es la creación de una estructura en "cascada", es decir, la combinación de clasificadores complejos, que rechaza el fondo de la imagen de entrada pasando más tiempo de cálculo en las áreas que puedan contener el objeto de interés.

A) *Características tipo Haar*:

El algoritmo de detección facial buscará características específicas que sean comunes en un rostro humano. Estas “características” son básicamente rectángulos blancos y negros.

Ilustración 22: características tipo Haar

B) *Imagen integral*:

En 2001, Paul Viola y Michael Jones, usaron el término “Imagen integral” dentro de su estructura de detección de objetos, para referirse a un método rápido y eficiente para calcular la suma de los valores de los píxeles en cualquier zona rectangular de una imagen dada.

C) *Adaboost*:

La idea principal es combinar la salida de algunos clasificadores débiles en una suma ponderada, creando así un clasificador final fuerte, cuyo error tienda exponencialmente a cero.

D) *Estructura en Cascada*:

El detector de Viola-Jones usa la técnica de Adaboost, pero organiza los clasificadores como una cascada de nodos de rechazo. Tan sólo el candidato que logre atravesar la cascada entera será clasificado como un rostro. De esta manera el coste computacional se reduce significativamente.

4.3.2. Algoritmos para el reconocimiento facial

a. Algoritmo EigenFace *[7]*

Este algoritmo se basa en la técnica PCA (Análisis de Componentes principales).

Esta técnica consiste en 2 fases, una de entrenamiento y otra de clasificación.

En la primera fase se forma un espacio a partir de las imágenes de entrenamiento. Este espacio es la matriz formada por los vectores propios. Éstos contienen la información de la variación de los valores de cada pixel del conjunto de imágenes.

Los primeros vectores tendrán la información más importante del espacio, mientras que a medida que recorremos la matriz la información pasa a ser redundante.

Por lo tanto, para formar el espacio no necesitaremos todos los vectores, lo que se traduce en que tendremos una reducción importante en la cantidad de información manejada.

Como estos vectores representan la variación de los píxeles de las imágenes, si representamos estos vectores como imágenes veremos que parecen "caras", de ahí que también se les conozca como eigenfaces.

Para finalizar la fase de entrenamiento, las imágenes que se emplearon al realizar el PCA se proyectan en el espacio.

Esta proyección caracteriza la imagen facial de un individuo como la suma de los diferentes pesos del espacio de imágenes.

En la fase de reconocimiento, una imagen facial desconocida es proyectada contra el espacio. Después, por medio de la distancia euclídea, se busca la imagen facial proyectada más parecida a la desconocida.

La técnica de PCA tiene dos defectos importantes.

Al estar basado en los valores de brillo de las imágenes, es muy sensible a variaciones en la iluminación, por lo que es importante que para utilizar esta técnica la iluminación esté controlada.

El otro defecto es que cuando se quieran añadir nuevas imágenes a las que conformaban el entrenamiento original, hay que realizar de nuevo el PCA y volver a proyectar todas las imágenes de nuevo.

b. Algoritmo FisherFace *[7]*

Otro método común en el reconocimiento facial es el LDA (Análisis Discriminante Lineal), o FisherFace.

En contraste con el anterior, este busca maximizar la varianza de las muestras entre personas y minimizarla entre muestras de la misma persona.

Con esto se logra obtener mejores resultados en caso de que haya variaciones de la iluminación y expresión respecto de las imágenes de entrenamiento.

Para lograr esto, en la fase de entrenamiento se necesita tomar varias imágenes de cada sujeto en diferentes condiciones de iluminación y pose.

De esta manera, la descomposición interpola o extrapola las demás condiciones de pose o iluminación que se pueden presentar aparte de las ya tomadas. La debilidad de la técnica está en que se necesitan varias tomas del mismo individuo y que estas sean representativas de las variaciones que se vayan a presentar en la aplicación real.

Además, esta técnica es más costosa computacionalmente que la de PCA.

El proceso de reconocimiento es básicamente el mismo que en el PCA. Lo que diferencia ambas técnicas es la forma de calcular dicho espacio.

Para calcular el espacio, se buscan dos matrices de covarianza, una que corresponde a las diferentes imágenes de una misma persona, y una que corresponde a las imágenes de diferentes personas.

La relación entre ambas matrices nos dará el subespacio LDA, compuesto por los vectores propios ya conocidos de la técnica PCA. Éstos son las fisherfaces que dan nombre a la técnica.

c. Algoritmo LBPH(Local Binary Patterns Histograms)

La idea del LBPH es no mirar la imagen entera como un vecto, sino describir sólo características locales de un objeto.

La idea básica de los patrones binarios locales es resumir la estructura local en una imagen comparando cada píxel con sus vecinos. Tomando un píxel como centro, si la intensidad del píxel central es mayor o igual a su vecino, entonces se denotará con 1 y 0 si no. Al final terminaremos con un número binario para cada píxel. Así que con 8 píxeles circundantes tendremos 2 ^ 8 posibles combinaciones, llamadas patrones binarios locales o códigos LBP. Un ejemplo de esto podemos verlo en la siguiente ilustración:

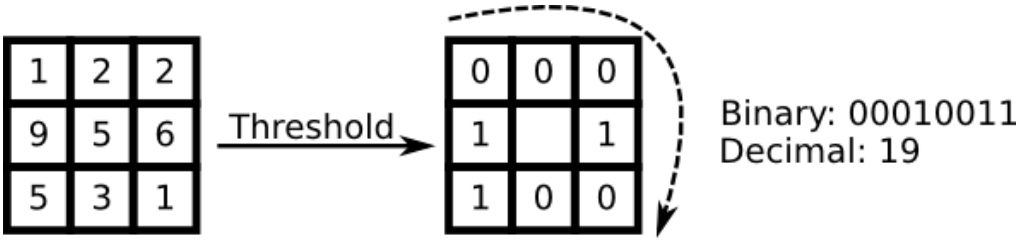

Ilustración 23: Código LBP

Por definición, el operador LBP es robusto contra las transformaciones monotónicas de escala de grises. Podemos verificar esto fácilmente mirando la imagen LBP de una imagen artificialmente modificada:

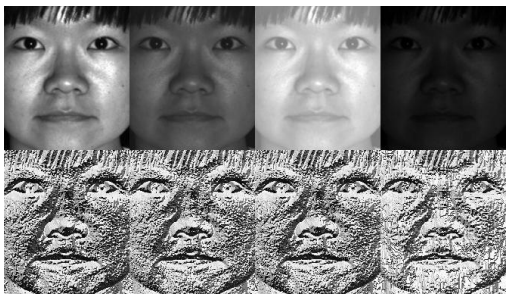

Ilustración 24: transformaciones LBP (ref)

4.3.3. Librería OpenCV

Los algoritmos anteriores no vienen implementados nativamente en java. Para poder utilizarlos tenemos dos opciones, implementarlos nosotros desde 0 o utilizar una librería externa que los contenga.

Una de las librerías que los implementa es OpenCV (Open Computer Vision). Ésta es

una librería desarrollada por Intel en 1999 y está bajo una licencia BSD. Actualmente van por la versión 3.2.0 y la versión estable es la 2.4.13 del 23 de Diciembre de 2016.

Es multiplataforma y contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estérea y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

En nuestro proyecto, utilizaremos la versión 3, puesto que en lo que concierne a los algoritmos de reconocimiento facial, tiene ciertas mejoras respecto a su antecesora.

Para poder utilizarlo en java, puesto que la librería nativa es en C y C++ habrá que compilar la librería en Linux para conseguir los ficheros .java y.so que podremos utilizar en nuestro proyecto haciendo una llamada a la librería nativa.

Hay que comentar que hay una diferencia en cuanto a esta compilación entre la versión 2 y la 3, puesto que en la 3 hay distintas funcionalidades que están en un paquete de módulos externos y en nuestro caso el modulo face, estará en ese paquete externo a la librería.

Este paquete, contiene los 3 algoritmos explicados anteriormente como sus métodos asociados para crear las comparaciones entre las imágenes del entrenamiento y las del reconocimiento.

4.3.4. Reconocimiento Facial

Ahora que sabemos los algoritmos existentes y la librería que nos los proporciona, vamos a analizar cómo se va a realizar este proceso de reconocimiento facial.

Lo primero que hay que entender, como se ha visto en los algoritmos, es que tienen que existir una serie de imágenes de muestra para los usuarios, las cuales tienen que tener un mismo tamaño para que los espacios dimensionales coincidan.

Es decir, para que el reconocimiento facial funcione, se debe dividir en 2 fases, una de entrenamiento y una de reconocimiento.

En la primera fase, la de entrenamiento vamos a recoger las imágenes y extraer los datos de las mismas para poder aplicar los algoritmos.

Para ello cada vez que se recoja una imagen, detectaremos el rostro que hay en ella, y se guardará una imagen nueva que contenga el rostro en una escala de grises y redimensionada a un tamaño específico para que todas sean iguales. Pero no hay que analizar todas las imágenes cada vez que se quiera hacer el reconocimiento. Cuando aplicamos el algoritmo a las imágenes, los valores propios y los vectores propios de cada una, se pueden guardar en un fichero XML. Este fichero solamente lo cambiaremos cuando haya nuevas imágenes.

En la segunda fase, la de reconocimiento, se recogerá una imagen nueva, como anteriormente, se detectará el rostro y se pasará a escala de grises. Después con los

métodos que nos proporciona OpenCV se compararán los datos extraídos de esta imagen nueva con los guardados en el fichero XML y nos dirá con qué imagen de las del entrenamiento tiene la mayor coincidencia.

4.3.5. Clases Java

Puesto que ya hemos indicado anteriormente el esquema de clases de la capa de persistencia como del modelo de datos, vamos a profundizar más en esta sección sobre la capa de Lógica de negocio.

Empezaremos con el paquete Servlets. Este contiene las distintas clases que nos llevarán a través de las páginas.

Las clases que tendremos son:

- **AltaCurso**: esta clase hace referencia a la parte de administración. Éste introducirá un nombre y una descripción para un nuevo curso, y enviará los parámetros a este servlet, que creará el objeto Curso y lo introducirá en la base de datos dándole un identificador.
- **BajaCurso**: el administrador tendrá una lista con todos los cursos. Este seleccionará el que quiere borrar y enviará como parámetro al servlet el identificador del curso. Con este se invocará al método baja() de la clase GestorBDcrs, que lo eliminará de la base de datos.
- **CierreSesion**: este servlet invalida la sesión que esté activa en ese momento.
- **EditarPwd**: éste es del usuario registrado, lo que hace es pasarle los parámetros de la contraseña antigua y la contraseña nueva, y si todo es correcto, llama al método modificarPwd() de la clase GestorBDusr.
- **EditarUsuario**: es igual que el anterior pero en vez de modificar la contraseña modifica el nombre del usuario.
- **LoginUsuario**: éste es el que nos creará la sesión del usuario. Para ello se le pasará el nombre y la contraseña, y si son correctos creará la sesión y meterá el objeto Usuario en ella.
- **Matricula**: éste dará de alta al usuario que hay en sesión en el curso que se le indique.
- **ModificarCurso**: se seleccionará un curso y al servlet se le pasará el identificador y los parámetros que se desea modificar, si alguno de ellos está en blanco o si el nombre nuevo coincide con otro curso, no se realizará la modificación.
- **RegistroUsr**: se introducirá un nombre de usuario y 2 veces la contraseña. Si el nombre es correcto, es decir que no esté en blanco y que no exista ya, y la contraseña introducida 2 veces coincide, se procederá a dar de alta al usuario y además crear la sesión de este para entrar en su zona de datos.
- **CancelarRegistro**: se le pasará el usuario que se ha metido en sesión y se eliminará de la base de datos.

Para el paquete de *logicaNegocio*, vamos a hacer uso de la librería OpenCV puesto que

realizaremos la recogida de imágenes y el entrenamiento facial.

Las clases que tenemos son:

- **DeteccionCaraTmp**: en su método principal, el único público, se le pasará la imagen que se quiere tratar. En él se hará la detección del rostro y sobre el que detecte, se hará un recorte a la foto en donde se ha detectado el rostro y esa sección se pasará a escala de grises. Esa foto se guardará en una carpeta que llamaremos tmp para después borrar la foto. Además se redimensionará para que tenga un tamaño de 125*150 pixels para que todas las imágenes sean del mismo tamaño, por lo que se explicó anteriormente en los algoritmos matemáticos.
- **DetectarCaraNueva**: básicamente es igual a la clase anterior, pero a esta se le pasan los datos del usuario para guardar los rostros en la carpeta del usuario y con un nombre en específico que será de la siguiente forma: img_IDusuario_nºImagen.jpeg. Con esto, se guardará la imagen en local.
- **EntrenamientoFacial**: en esta clase tenemos 3 métodos, puesto que hay 3 algoritmos matemáticos. Todos ellos tienen el mismo funcionamiento, lo único que cambiará es la forma en que se va a realizar el procesado del algoritmo, es decir que solo cambiará un método que concordará con el algoritmo aplicado.

  Hemos hecho esto así para llamar solamente al entrenamiento del algoritmo que queramos.

  El funcionamiento de estos métodos es el siguiente:

  - Se buscan las imágenes del usuario
  - Se hace una matriz con los identificadores del usuario y una lista de matrices que cada matriz será cada una de las imágenes. Es decir la primera entrada de la matriz de identificadores concordará con la primera matriz de la lista de matrices.
  - Creamos uno de los objetos siguientes:

    ```
    BasicFaceRecognizer fr= org.opencv.face.Face.createEigenFaceRecognizer();
    BasicFaceRecognizer fr=org.opencv.face.Face.createFisherFaceRecognizer();
    LBPHFaceRecognizer fr= org.opencv.face.Face.createLBPHFaceRecognizer();
    ```

  - La matriz de identificadores y la lista de matrices, se pasan al método `fr.train(Identificadores,lista)`. Con esto tendremos los datos biométricos de todas la imágenes, pero para hacerlo persistente, hay que ejecutar el método `fr.save(new File("ficheroBio.xml"))`.

En el paquete api.ajax se recogerán las clases que se llamarán desde el JavaScript con AJAX y nos devolverán una respuesta a través de JSON. Estas clases son:

- **Entrenamiento**: esta clase le llegará como parámetro el curso que se quiere entrenar, y este se lo pasará a la clase *EntrenamientoFacial*. Si esta ha creado el fichero correctamente, JSON enviará un mensaje diciendo que el entrenamiento se ha ejecutado correctamente y sino, uno de error.
- **GuardarFoto**: esta clase recogerá la imagen enviada por parámetro en Base64 y

se decodificará creando un objeto File. Este objeto, junto con los datos del usuario que está en la sesión se le pasarán a la clase *DetectarCaraNueva* y si se guarda en local la imagen correctamente, se insertará un registro en la Base de Datos. Además cuando todo es correcto se devolverá por JSON un mensaje diciendo que la imagen se ha guardado, y en caso contrario un mensaje diciendo que no ha podido realizarse la acción.

- **ReconocimientoFacial**: esta clase es la más importante de todas. Su funcionamiento es el siguiente:
    - Primero creamos uno de los objetos fr con `BasicFaceRecognizer` o `LBPHFaceRecognizer.`
    - Con este objeto, cargamos el fichero xml que hemos creado en la fase de entrenamiento. Para ello utilizamos `fr.load("ficheroBio.xml").`
    - Ahora, como en la clase de *GuardarFoto*, creamos un File a partir del parámetro codificado en Base64.
    - Este File se le pasará a la clase DeteccionCaraTmp, y cuando haya creado la imagen nueva del rostro en la carpeta tmp, borraremos el file inicial puesto que ya no es necesario.
    - Ahora a partir de la imagen nueva creamos una matriz y la pasamos también a escala de grises. Este paso es importante porque aunque ya esté en ese formato, es imprescindible volver hacerlo para que funcione.
    - Después se crearán un int[] y un double[] que se pasarán al método fr.train(matiz_imagen, int[],double[]) . Este método introducirá en int[] el identificador de la imagen con la que más coincidencia tiene y en double[] el parámetro confidence, que posteriormente se podrá analizar para comprobar el porcentaje de acierto en la identificación.
    - Teniendo estos 2 datos, los devolveremos a través de JSON para mostrarlos por la interfaz gráfica.
- **UsuariosCurso**: esta clase nos devolverá por JSON en número de usuarios que están matriculados en un curso determinado. Esta información se verá en la página de información del curso.
- **VerImagenes**: esta clase, recogerá de la base de datos la lista de imágenes del usuario. Después cada imagen se pasará a base64 y se añadirán en otra lista que será la que se devuelva a través de JSON para poder mostrar estas imágenes en el html.
- **NumImgUsr:** nos devuelve el número de fotos que tiene el usuario guardadas para su reconocimiento.
- **GuardarFotoAuto:** se le pasará un usuario y una lista de String que contendrá imágenes en Base64. Este servlet procesará todas las imágenes, las guardará y después si todo ha ido correctamente hará el entrenamiento de las fotos. Nos devolverá un JSON diciendo si el proceso se ha realizado de manera satisfactoria o no.

4.4. Interfaz gráfica

Para implementarla más rápidamente, vamos a ayudarnos de la herramienta Bootstrap que contiene css ya creados y demás herramientas para el diseño gráfico, con el que ahorraremos tiempo y nuestra página tendrá un diseño más cuidado.

La zona común estará compuesta por estas 4 páginas:

Ilustración 25: Página de inicio

En el catálogo de cursos vamos a tener una tabla con todos los cursos existentes y 2 botones, uno para ir a la descripción del curso y otro para matricularnos en él. Si no hay iniciada una sesión cuando pulsemos sobre el botón de matricularse, nos enviará a la zona de Inicio de sesión.

Ilustración 26: Página de catálogo de cursos

En la zona de inicio de sesión tendremos 2 zonas el formulario de entrada y un botón para llevarnos a la zona de creación de un usuario nuevo.

Ilustración 27: Página inicio de sesión

En el registro, tendremos que introducir un nombre de usuario y una contraseña que tendremos que repetir para mayor seguridad.

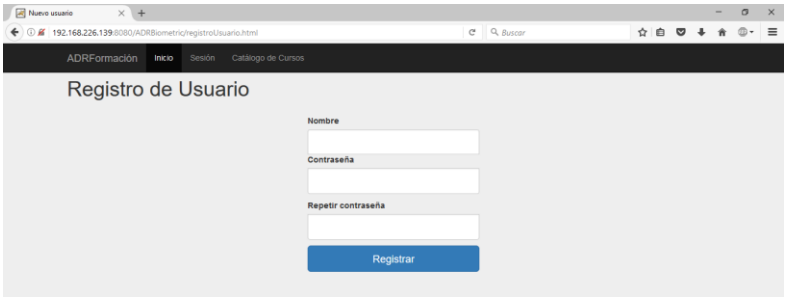

Ilustración 28: Página registro de usuarios

Antes de acceder al perfil creado, primero debemos hacernos fotos, esto se puede ver en la ilustración inferior, y podemos comprobar que hasta que no haya un mínimo de fotos no se activará el botón de terminar que nos llevará al perfil. Si no disponemos de cámara para realizar las fotos podemos darle a cancelar y se cancelará el proceso de registro.

Ilustración 29: Página de recogida de imágenes en el registro

La zona del usuario registrado tendrá estas partes:

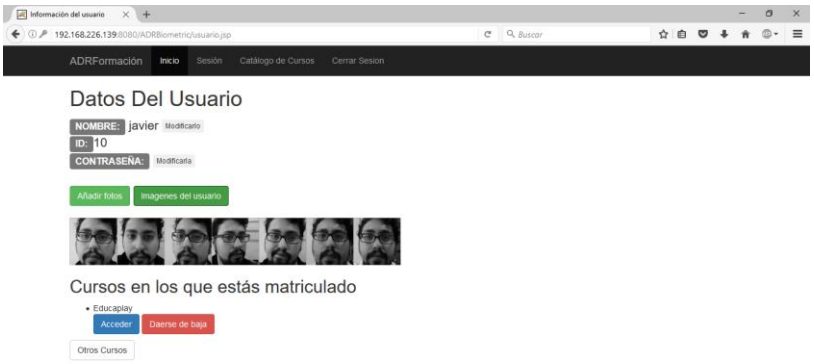

Ilustración 30: Página perfil del usuario

Una zona de datos del usuario, que contendrá el nombre, el identificador y la subzona de las imágenes. Si pinchamos sobre Imágenes del usuario, vemos que debajo aparecen las fotos que se han recogido hasta el momento.

Además si pinchamos sobre modificar en contraseña o en nombre tendremos lo siguiente:

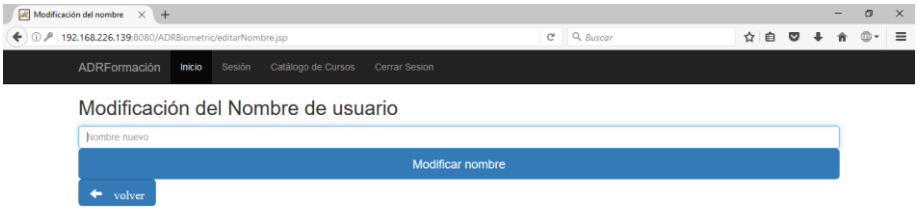

Ilustración 31: Página modificación nombre usuario

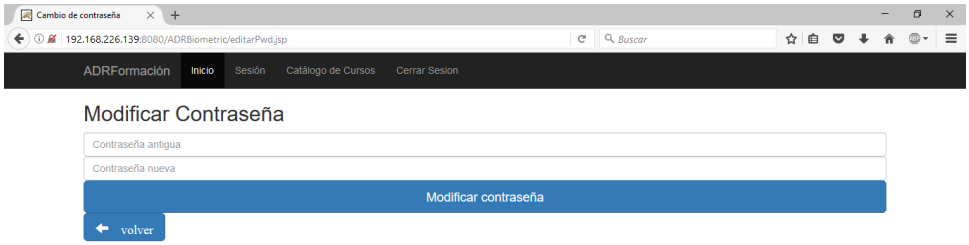

Ilustración 32: Página modificación de contraseña

Si queremos hacernos más fotos pulsamos sobre Añadir fotos y tendremos la siguiente pantalla de recogida de imágenes:

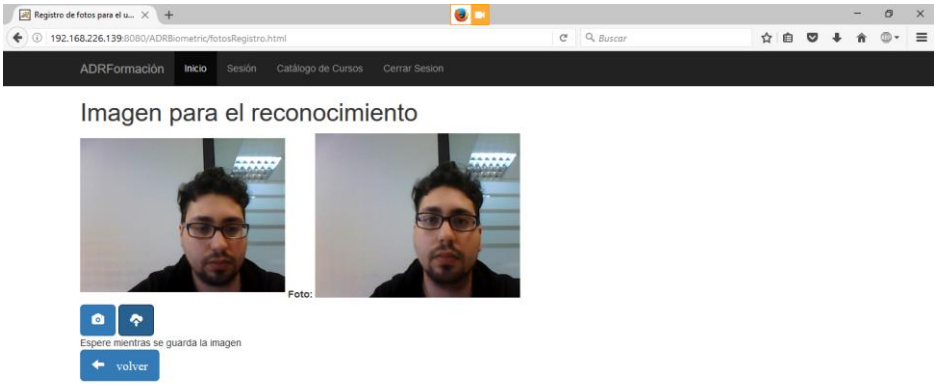

Ilustración 33: Página captura de imágenes

Con el botón del símbolo de la cámara se realiza la instantánea de la derecha, y con el botón de su derecha, se envía al servidor para que la trate y guarde el rostro de la imagen si lo detecta.

Ilustración 34: Imágenes pertenecientes al usuario Javier

Como podemos apreciar, la imagen img_10_8.jpeg es la que se ha recogido en la Ilustración 33.

Si damos al botón Acceder de un curso de la página del perfil del usuario, encontraremos lo siguiente:

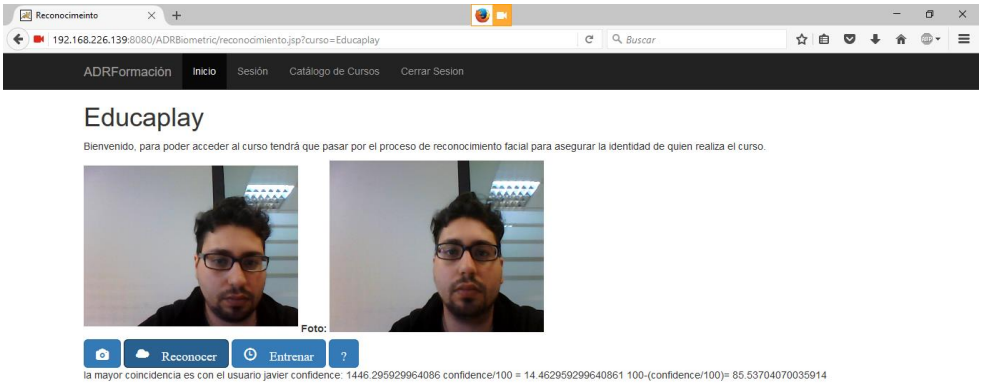

Ilustración 35: Página de reconocimiento para acceso al curso

El resultado que se encuentra al pie, es resultado de pulsar el botón Reconocer. Aunque en el diseño esté distinto, en la interfaz final en el resultado solo se verá el resultado de uno de los algoritmos. Además tampoco devolverá el nombre del usuario, puesto que, en el diseño final el XML con los datos de entrenamiento, solo tendrá las imágenes de un usuario para realizar la comparación, y por tanto solamente sacaremos el resultado del algoritmo finalmente utilizado.

La zona del administrador se ve de la siguiente forma:

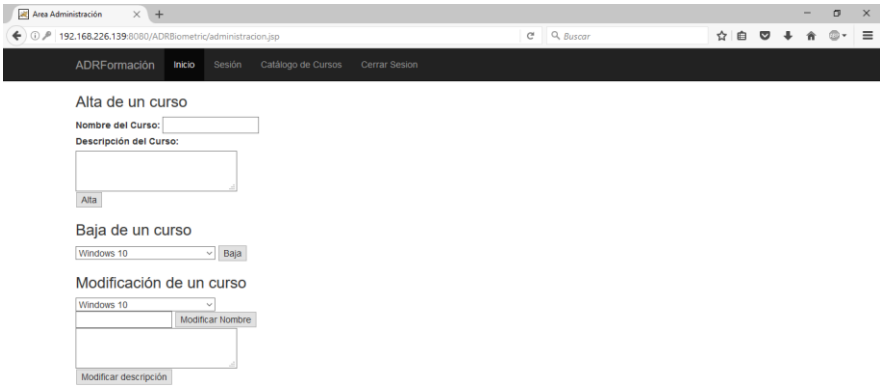

Ilustración 36: Página de administración de cursos

Tiene 3 zonas diferenciadas:

- Alta de un curso: en ella hay un formulario en el que introduciremos en nombre y la descripción de un curso nuevo, y cuando demos en el botón *alta*, se registrará en la base de datos.
- Baja de un curso: el desplegable contendrá todos los cursos del catálogo. Cuando hayamos seleccionado uno, al pulsar sobre el botón *baja*, lo eliminaremos de la base de datos.
- Modificación de un curso: el desplegable contendrá todos los cursos del catálogo. Cuando hayamos introducido un nombre al pulsar en *Modificar nombre*, si todo es correcto se modificará el nombre en la base de datos, y lo mismo ocurrirá si hacemos la parte de la descripción.

4.5. Integración con la plataforma

Para la integración del servicio en la plataforma hay que tener en cuenta varias cosas:

1. La aplicación java que hemos creado funciona cogiendo datos de la sesión.
2. La sesión de java y la de PHP no funcionan de la misma forma.
3. La base de datos no es exactamente la misma.
4. CORS (Cross Origin Resource Sharing).

Con esto en mente, tendremos que cambiar un par de cosas de la aplicación, puesto que la distinta forma del traro de sesión nos limitará bastante y las bases de datos, la de la plataforma y la del servicio, solo tendrán en común el nombre del usuario.

Estaba previsto que esta integración se hiciera en un servidor en la nube después de realizarla en uno local y comprobar que todo funcionara. Por falta de tiempo no se ha podido integrar en el servidor de Amazon, pero ha quedado todo documentado para su futura implementación si así se desea.

4.5.1. Sesión

En principio habíamos utilizado la sesión en nuestra aplicación puesto que simulábamos una plataforma completa. Pero para solamente dar servicio a los métodos de recogida de fotos, entrenamiento y reconocimiento, no vamos a necesitar una sesión sino pasarle por parámetro que usuario es al que se quieren aplicar éstos mismos.

Por ejemplo, para ver un poco cómo podríamos hacer esta integración vamos a ver el método VerImagenes que devuelve la lista de imágenes del usuario.

Primero vemos el código original:

```
@Override
protected void doGet (HttpServletReques request, HttpServletResponse response)
        throws  ServletExecption, IOException{
        GestorBDutils gutls= new GestorBDutils();
        HttpSession ses = request.getSession();
        Usuario u= (Usuario) ses.getAttribute( “usuario” );
        int idu= u.getID();
        List<Imagen> limgs;
        try{
                imgs= gutls.imagenesUsuario(idu);
                List<Img> lista =new ArrayList<Img>();
                for(int I=0; i<imgs.size();i++){
                        File fimg= imgs.get(i).getDatosImg();
                        String nomImg= fimg.getName();
                        String s= "" + Imagen.fileToBase64(fimg);
                        Img im= new Img(nomImg, s);
                        lista.add(im);
                }
                String json = new Gson().toJson(lista);
                response.setContentType( "application/json" );
```

```
                response.getWriter().write(json);
        } catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
        }
}
```

En el primer recuadro, vemos que el objeto usuario lo saca de la sesión y luego de ese objeto, saca el Identificador del usuario.

Por ultimo vemos la respuesta que enviará a través de JSON que es una lista con datos de las imágenes.

Para poder hacer la integración, solo habría que cambiar 2 cosas. Lo primero es pasar un parámetro con el nombre de usuario y lo segundo es cambiar el método de doGet a doPost.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
GestorBDutils gutls= new GestorBDutils();
GestorBDusr   gusr = new GestorBDusr();
String nomUser = request.getParameter("usuario");
Usuario u= gusr.dameUsr(nomUser);
int idu= u.getID();
List<Imagen> imgs;
try {
        imgs = gutls.imagenesUsuario(idu);
        List<Img> lista= new ArrayList<Img>();
        for(int i=0; i< imgs.size();i++){
            File fimg= imgs.get(i).getDatosImg();
            String nomImg= fimg.getName();
            String s=""+ Imagen.fileToBase64(fimg);
            Img im= new Img(nomImg, s);
            lista.add(im);
        }
        String json = new Gson().toJson(lista);
        response.setContentType("application/json");
        response.getWriter().write(json);
} catch (ClassNotFoundException | SQLException e) {
        // TODO Auto-generated catch block
        System.out.println(e.getMessage());
}
}
```

Ilustración 37: código con parámetro

Como podemos ver, solamente hemos cambiado el método y dentro de él, en vez de sacar el objeto usuario de la sesión, lo creamos con el nombre de usuario que le pasamos por parámetro. Todo lo demás seguiría siendo igual.

Para la llamada desde Javascript con AJAX, sería de la siguiente forma:

```
var contenedor= document.getElementById("imgs");
$.ajax({
    type : "POST",
    url : "api/VerImagenes",
    data: {usuario: nom_User}, --> nom_User contendrá en nombre de usuario
    success: function (data) {
                var datos= "";
                for(var i=0; i<data.length;i++){
                    datos +="<img alt='"+data[i].nombre+"' src='"+data[i].datos +"' width='80' height='80'>";
                }
                contenedor.innerHTML= datos;
    }
});
```

Ilustración 38: método Ajax

Pasamos 3 instrucciones al método AJAX:

- type : tipo de envío
- url : url del método que se invoca
- data : parámetros que se le pasan al servicio

Además de esto en success vamos a tratar lo que nos devuelva el método invocado.

4.5.2. Base de datos

Como hemos podido comprobar, la tabla más importante, en cuanto a integración se refiere, es la de Usuario. Esta solamente contendrá un identificador, un nombre de usuario y una contraseña. En la base de datos final para la integración, la contraseña desaparecerá puesto que éste campo está solamente para la recreación de una plataforma completa en la que los usuarios se registran y se loguean.

Es decir, la base de datos final en forma de diseño lógico tendría esta forma:

Usuario

ID_u	nom_usr

Imagen

ID_img	ID_u	nom_img	DatosImg

CE: Usuario

Biometria

ID_b	ID_u	nom_xml	contenido

CE: Usuario

Así en el servicio sólo vamos a tratar con nombres de usuarios, con los datos de las imágenes y los ficheros de biometría, asegurando así que estos datos sean independientes de la base de datos de la plataforma.

La única relación entre las bases de datos va a ser el campo nom_usr de la tabla usuario.

4.5.3. CORS (Cross Origin Resource Sharing)

Puesto que vamos a tener 2 dominios distintos, el servidor donde alojamos la aplicación y el servidor desde donde hacemos las llamadas al mismo, vamos a necesitar hacer solicitudes de orígenes cruzados. Por razones de seguridad, esto está restringido a que

solo se puedan hacer peticiones desde el mismo dominio.

Para poder permitir estas solicitudes CORS, dentro del servidor TomCat vamos a tener un filtro que hará que se pueda realizar este proceso. Por defecto no viene configurado.

Para poder activarlo, dentro del web.xml del servidor TomCat, vamos a escribir lo siguiente:

```
<filter>
	<filter-name>CorsFilter</filter-name>
<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
</filter>

<filter-mapping>
	<filter-name>CorsFilter</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

Con estas líneas, vamos a permitir el CORS y por tanto podremos hacer las peticiones AJAX a nuestro servidor de la aplicación desde cualquier punto.

Además de esto, también podemos restringir el CORS a un dominio en concreto si cambiamos el url-pattern y en vez de /* ponemos un dominio como http://www.ejemplo.com/ y con ello solo aceptaría peticiones desde ese dominio.

En nuestro caso hasta el momento se quedará con /*.

Además de esta integración, en el Anexo D, podemos ver un API de los métodos AJAX y la compatibilidad de la tecnología WebRTC que nos permite activar la cámara web desde un navegador.

También podemos ver cómo, además de navegadores de escritorio, se puede acceder a la aplicación desde dispositivos móviles.

5. Pruebas

Para la realización de las pruebas vamos a coger a 2 usuarios y vamos a comprobar cómo se comporta uno de los algoritmos (LBPH).

Se ha elegido éste porque Eigenface es muy sensible a cambios de luz y posición, y Fisherface necesita muestras con distintos tipos de iluminación y poses, y puesto que queremos que la recogida de la muestra sea automática para el usuario, no nos sirve.

Para ello iremos comprobando qué valores nos devuelve el algoritmo a la hora del reconocimiento cuando tenemos 1, 5, 10, 20 y 30 fotos por cada usuario.

En las tablas del ***Anexo C*** vamos a ver, respecto a los usuarios Javier y Juan, cual es el valor devuelto por el algoritmo en diferentes posiciones o características. También hay que tener en cuenta que el usuario Javier lleva gafas y el usuario Juan no. Por tanto es una característica a tener en cuenta.

En la **tabla 1 del *Anexo C***, al recoger 5 imágenes por usuario, vamos a coger también las distintas características como las gafas, la boca, la lengua o una imagen con distinta iluminación, etc.

En los resultados de 1 foto, no hemos seguido haciendo pruebas, puesto que los valores devueltos se nos solapan entre una foto a contra luz y una foto normal del otro usuario, es decir, que salen valores superiores en contra luz que en la normal.

Los valores obtenidos en la **tabla 1 del *Anexo C*** para 5 fotos por usuario, vemos que cuando es el mismo usuario, en Javier varía de 55 a 76, y para Juan tenemos un intervalo de 56 a 69. Los valores más altos son de la imagen a contra luz.

Cuando son 2 usuarios distintos, es decir, cuando comparamos los datos biométricos de Javier con una foto que se hace Juan en ese momento, vemos que el rango va de 83 a 94 y cuando es al contrario, tenemos un rango que va de 70 al 86.

Hay que apuntar que la muestra de imágenes en Javier, es más variada, y sin embargo la muestra de Juan es más uniforme.

En la **tabla 2 del *Anexo C*** podemos ver los resultados para 10 imágenes por usuario. Como en la anterior, las imágenes tendrán distintas poses.

También veremos los resultados cuando el usuario tiene 20 fotos para su identificación.

Los resultados obtenidos nos dicen:

- En la tabla de 10 fotos por usuario:
    - Cuando tenemos el mismo usuario tenemos un rango de 57 a 79.
    - Cuando son distintos un rango de 72 a 86.

  Como podemos observar, los rangos se vuelven a solapar como en el caso de 1 foto por usuario. Por tanto una muestra de 10 fotos no nos sirve.
- En la tabla de 20 fotos por usuario:
    - Con el mismo usuario tenemos rangos de 45 a 71 y de 54 a 61. Dentro

del primero tenemos en realidad un rango de 45 a 56 y un valor extremo de 71 de la foto a contraluz.

- Cuando son distintos tenemos rangos de 74 a 85 y de 74 a 82.

Hasta el momento, estos resultados son los mejores, y los más uniformes, exceptuando el valor 71 de la foto a contraluz.

Como podemos ver en la **tabla 3 del *Anexo C*** aunque hemos subido el número de fotos por usuario, los rangos obtenidos son bastante parejos a los obtenidos con los de 20 fotos por usuario.

- Cuando es el mismo usuario: tenemos rangos de 46 a 72 y de 52 a 67, siendo los valores altos los de la foto a contraluz.
- Cuando son distintos usuarios: tenemos rangos del 72 al 87 y de 67 al 87. Vemos como estos se solapan con los valores más altos de las fotos a contraluz de cuando es el mismo usuario.

La conclusión de las pruebas hasta el momento, es que con 20 fotos, nos salen valores más óptimos.

Ahora, sabiendo que con 20 fotos es lo más óptimo hasta el momento, pasaremos a realizar las pruebas que nos quedan con este número de fotos por usuario.

Lo primero que hay que decir, es que con los resultados obtenidos hasta el momento, vamos a poder establecer 3 rangos diferenciados en cuanto al reconocimiento:

- Valores entre 40 y 60: cuando obtenemos un valor entre este rango, podremos decir que el usuario es el correcto.
- Valores entre 60 y 70: no se puede decir si es o no el usuario correcto, se pedirá entonces intentar reconocer de nuevo al usuario.
- Valores mayores de 70: cuando obtenemos un valor superior a 70 podremos decir que el usuario no es correcto, es otra persona.

Hemos hecho esta clasificación puesto que es mejor que salga un falso negativo que un falso positivo, es decir, mejor que diga que no somos nosotros aunque lo seamos, a que diga que otra persona somos nosotros.

La primera prueba que vamos a realizar es, teniendo 20 fotos del usuario Juan, vamos a intentar pasarnos por él con el usuario Javier tomándonos fotos con distintas características. La principal será que el usuario Javier se haga fotos sin gafas, puesto que Juan no tiene. Éstos resultados se pueden ver en la **tabla 4 del *Anexo C***.

Podemos comprobar que solo 1 vez hemos conseguido bajar de 70, y este ha sido el caso de ponernos a la máxima distancia posible para detectar el rostro y además sin gafas, para engañar al sistema. Pero como el valor obtenido está en el rango de incertidumbre, hemos conseguido que aunque no esté en la zona de asegurar que no somos el usuario, tampoco diga que lo es.

En la **tabla 5 del *Anexo C*** vamos a intentar comprobar que reconoce de verdad al usuario Javier. Para ello este usuario intentará reconocerse en distintas situaciones y obtener los resultados, para saber así si de verdad el rango de acierto es óptimo.

Dados los resultados, la mayor parte de las imágenes sí reconocen al usuario, es decir que el valor es inferior a 60, puesto que son frontales y con la misma iluminación. Las que están en el rango de incertidumbre, entre 60 y 70, son las imágenes que no son de frente o hay un pequeño cambio de iluminación como es el caso de la foto con un cristal de fondo. Además hay un falso negativo, puesto que en la foto sin gafas y tomada desde la mayor distancia posible para que detecte nuestro rostro, el valor obtenido es de 74.

Todas estas pruebas se han realizado en el mismo escenario, por tanto vamos a intentar realizar estas pruebas en otro distinto, para saber si así funciona también nuestro rango de acierto.

Como podemos observar en los resultados de la **tabla 6 del *Anexo C***, aparecen varios resultados con más de 60 pero sin llegar al máximo de 70, exceptuando uno. Esto es resultado de que la iluminación, aunque es parecida, no es totalmente la misma. Y por tanto el resultado del LBPH se eleva en todos los casos. De los resultados tenemos que 14 de los 24 casos están en el rango de menos de 60 y los otros 9 están en el rango de incertidumbre y uno en el rango de más de 70. También hay que decir que este resultado de más de 70 es de una foto tomada desde arriba y aunque detecte la cara, no es totalmente frontal como las de la muestra tomada para el entrenamiento.

La pregunta que nos planteamos ahora mismo es, ¿hay que aumentar el rango de acierto y acotar más el de incertidumbre o hay que mantener los rangos como los tenemos hasta el momento?

Si queremos seguridad, lo mejor es dejar los rangos como están puesto que es mejor tener falsos negativos.

La desventaja es que el usuario va a necesitar estar en unas condiciones muy parecidas siempre que quiera realizar el reconocimiento. Es decir, cada vez que quiera utilizar este servicio, va a tener que intentar recrear las condiciones de luz que tenía cuando se realizó la muestra de imágenes de entrenamiento.

Además del estudio realizado, vamos a probar cómo se comporta el reconocimiento de un usuario, cuando en vez de estar sus parámetros biométricos aislados, están junto con los de otros usuarios.

Para ello nos creamos un usuario y hacemos también 20 fotos como hicimos en las pruebas anteriores.

Además añadiremos un campo más a la **tabla 7 del *Anexo C***, puesto que en estas pruebas además del valor nos devolverá si la foto con la que más coincide es del usuario jasantf.

Como podemos comprobar en la tabla, todas las fotos nos identifican por usuario, aunque si nos fijamos en los valores, la mayoría están cercanos a 60 o sobrepasan este valor, que si recordamos, era el límite del rango para que nos identificara cuando teníamos los datos biométricos separados.

También observamos que la foto a contra luz, aun sacando un valor de 81 sigue diciendo que el usuario es jasantf.

Además de esto, el coste computacional es mayor, puesto que tiene que buscar la

coincidencia entre todos los datos que existen en el fichero xml, y por tanto esto se ha reflejado en la respuesta. En las anteriores pruebas, el resultado lo devolvía en unos 2 segundos. Recordemos que tiene que detectar, recortar y redimensionar la imagen para poderla comparar con los datos. Ahora con 6 usuarios distintos con sus respectivas fotos, está tardando unos 4 segundos en devolver la respuesta, es decir unos 2 segundos en tratar la imagen más otros 2 segundos en realizar la búsqueda.

Si con 6 usuarios, está tardando el doble, cuando el fichero crezca y tenga miles de usuarios, la respuesta va a tardar mucho y por tanto este funcionamiento será inviable.

Por tanto no seguiremos haciendo más pruebas.

6.Seguimiento y control

En el siguiente apartado compararemos a través de una tabla las horas que se habían planificado en el apartado 1.7 con las horas reales que ha ocupado la realización del presente trabajo.

Tarea	Horas planificadas	Horas reales
Total Memoria	**37 h**	**36h**
Planificación	8 h	10 h
Desarrollo	22 h	24 h
Introducción y resumen	3 h	1 h
Opinión personal	2 h	30 min
Lecciones aprendidas y bibliografía	2 h	30 min
Reuniones	**8 h**	**6 h**
Pruebas	**10 h**	**20 h**
Análisis	**10 h**	**15 h**
Total Diseño	**33 h**	**32 h**
Diseño BD	5 h	3 h
Diseño Clases Java	20 h	17 h
Diseño Interfaz Gráfica	8 h	12 h
Total Implementación	**202 h**	**221 h**
Implementación BD	30 h	46 h
Implementación Clases Java	130 h	145 h
Implementación Interfaz Gráfica	42 h	30 h
TOTAL	**300 h**	**330 h**

Tabla 17: Horas totales del proyecto

Como podemos apreciar, donde más horas se han añadido a las planificadas es en el apartado de implementación. También se aprecia que en la parte de pruebas, se ha aumentado el doble de horas, de las 10 planificadas a las 20 horas reales. Esto es debido a que se han realizado pruebas tanto en la plataforma completa como en la fase de integración.

7.Conclusiones

Aunque los sistemas biométricos estén en auge, todavía hay mucho camino por recorrer, puesto que el reconocimiento facial no está tan avanzado como otros sistemas biométricos como son la huella dactilar o el escáner de retina.

Dentro de nuestro estudio hemos visto que el reconocimiento facial a través de cualquier cámara web, está ligado fuertemente a las condiciones de iluminación y pose que se recogen en la muestra del entrenamiento facial. Cuando el usuario sale de esas condiciones, al sistema le cuesta reconocer a la persona.

También hay que apuntar que normalmente todos los datos obtenidos de las muestras de cada usuario están en un solo fichero y que es a partir de éste desde donde se buscan las coincidencias.

Sin embargo, nosotros hemos realizado el estudio teniendo los datos de cada usuario aislados y que cuando se intente reconocer a un usuario solamente se compare con sus datos.

Con ello hemos podido establecer unos márgenes de acierto y error cuando el usuario intenta reconocerse frente a sus datos, y cuando intenta reconocerse frente a unos datos que no son suyos.

Con estos rangos podemos tomar una decisión final sabiendo si podemos fiarnos o no del resultado obtenido.

Además de esto, el coste computacional desde que enviamos la petición hasta que nos devuelve un resultado es de aproximadamente 2 segundos, mientras que, como hemos podido comprobar, en cuanto tenemos un fichero con todos los usuarios, el coste computacional aumenta y por tanto la respuesta tarda bastante en devolverse.

8.Lecciones aprendidas

Durante el proyecto se han aprendido varias cosas. Además de toda la información sobre biometría y las técnicas que existen, se ha aprendido a buscar información sobre sistemas de reconocimiento facial de código abierto e intentar encontrar como trabajan estos.

Con ello pudimos saber que la mayoría de ellos utilizaban la librería OpenCV y con ello empezar a estudiar esta librería que es la base de nuestro sistema.

Además también hemos aprendido como poder utilizar librerías que son de C y C++ en un lenguaje como Java. Esto es posible gracias a JNI y ANT que nos permiten compilar estas librerías y crearnos 2 ficheros (.jar y .so) que utilizaremos como librerías Nativas.

Al principio esto nos dio muchos problemas puesto que en OpenCV los métodos que necesitábamos para el reconocimiento facial estaban en un paquete externo.

Cuando solucionamos esto, nos surgió otro problema, que, aunque se creaba el .jar y

el .so, dentro no se encontraban los métodos del reconocimiento.

Esto era así porque dentro del fichero *CMakeLists.txt* del módulo que contenía a éstos le faltaba una instrucción para que también los compilara y los añadiera dentro del .jar y .so.

Con todo esto solucionado, el siguiente problema a solucionar fue la recogida de la imagen a través del navegador. Investigando un poco encontramos que a través de html5 y JavaScript se podía realizar una captura de la cámara y que el contenido de esta lo ponía codificado en base64 dentro de un canvas. Esto se puede hacer a través del API WebRTC. Ésta, está más desarrollada en el ANEXO D.

Al tener métodos de decodificación de Base64 en java el envío de la imagen desde el navegador al servidor fue fácil, puesto que enviábamos el contenido como un parámetro en texto plano.

Además hemos aprendido a adecuarnos al entorno de trabajo para el que necesitábamos desarrollar el sistema.

Esto es así puesto que como ADR trabaja en sus plataformas en PHP, hemos tenido que crear un sistema conectable a esta tecnología, puesto que, OpenCV no está disponible a día de hoy para esta tecnología.

Por tanto hemos tenido que utilizar nuestros conocimientos adquiridos durante la carrera para poder crear un sistema compatible.

Y para ello utilizamos Java que es compatible con la librería y AJAX a través de JavaScript que es compatible con PHP. Así, además de compatibilidad hemos conseguido que nuestro sistema se pueda conectar a cualquier plataforma web.

Además hemos aprendido a adaptarnos a los cambios propuestos por el cliente. Esto se debe a que se decidió cambiar el sistema de tener a todos los usuarios en un mismo fichero biométrico a tener cada uno por separado para que la eficiencia de la respuesta fuera más rápida.

9.Opinión personal

En mi opinión los sistemas de biometría facial son el futuro, puesto que a día de hoy la mayoría de los dispositivos, ya sean portátiles, móviles o tablets, llevan integrada una cámara con la que podemos tomarnos una foto para reconocernos.

Aunque todavía hay que depurarlos puesto que, como hemos visto en el estudio, a día de hoy el reconocimiento está fuertemente ligado a las condiciones de luz y pose de la muestra.

Aunque lo hemos desarrollado dentro de la aplicación, no hemos llegado hacer pruebas con el algoritmo Fisherface debido a que se quería que la muestra obtenida por el usuario fuera automática, es decir, que se recogieran todas las imágenes de una sola vez. Esto para Fisherface no es óptimo, puesto que este algoritmo necesita una muestra con imágenes de la persona en distintas condiciones de luz y pose.

Por tanto, en mi opinión esto sería un tema a investigar o hacer pruebas en el futuro

para saber cómo responde el algoritmo y si da mejores resultados que LBPH.

Otro de los puntos que se nos han quedado en el tintero es el reconocimiento a través de vídeo.

Durante la investigación en el periodo de prácticas, vimos que era posible, pero era dentro de una aplicación local que conectaba la cámara del equipo en el que estaba el programa, y esto no nos servía para un servicio en red.

Aun así hay técnicas para poder sacar fotogramas de un vídeo, y por tanto, sería posible, en vez de enviar una imagen estática al servidor, enviarle un vídeo y obtener la muestra de rostros a partir de los fotogramas del video.

10.Bibliografía

[1] OpenCV página oficial del proyecto (http://opencv.org/)

[2] Apuntes de la asignatura Programación de aplicaciones web, *Autor: Francisco J. García Izquierdo*

[3] Apuntes de la asignatura Programación de bases de datos, *Autor: Francisco J. García Izquierdo*

[4] Apuntes de la asignatura Diseño de bases de datos, *Autores: Arturo Jaime Elizondo y Cesar Domínguez Pérez*

[5] Implementación del algoritmo de detección facial de Viola-Jones, *Autor: Joaquín Planells Lerma* (ref.)

[6] Aplicación para Detección y Reconocimiento, Facial en Interiores *Autor: Pătraşcu Viorica Andreea* (ref.)

[7] PFC Reconocimiento facial, *Autor: Juan Alfonso Urtiaga Abad* (ref.)